TECHNIQUES FOR INCREASING RELIABILITY AND SCALABILITY OF LIVE STREAMING SYSTEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Maya Haridasan

August 2008

TECHNIQUES FOR INCREASING RELIABILITY AND SCALABILITY OF LIVE STREAMING SYSTEMS

Maya Haridasan, Ph.D.

Cornell University 2008

Media content distribution is a widely useful tool that has seen increasing popularity driven by the rise of new technologies. File sharing constitutes one popular style of content distribution, whereby users are interested in downloading all data as fast as possible. File-sharing systems are efficient in distributing data that is fully available in advance and when there are no strict time or ordering constraints on the reception of data by interested nodes. This dissertation focuses on a second style of content distribution, in which data must be delivered at a constant rate, with minimal latency from the original time of distribution by the sender. Live streaming systems have become quite popular in recent years, and in places like China, are now widely used for broadcasting television channels to several thousands of users.

The peer-to-peer paradigm (P2P) allows live streaming systems to scale on the number of users and to be robust in the face of simple failures. By employing interested nodes' upload capacities to help upload data to other nodes, the costs at the sender are minimized and the system is able to scale to very large numbers of receivers. Unfortunately, the reliance on untrusted nodes' resources also leads to security problems and lack of guarantees, leaving such systems vulnerable to attacks that may impede nodes from receiving data in which they are interested. In particular, live streaming protocols deserve special attention

since their time sensitive nature makes them more susceptible to the packet loss rates induced by attacks.

In this dissertation, I propose and evaluate techniques that enhance the scalability and resilience of live streaming systems. First, I investigate the vulnerabilities of previously proposed tree-based and mesh-based dissemination systems and present SecureStream, a P2P live streaming system built to tolerate malicious behavior at the end level. I present the main components of SecureStream and present results that demonstrate its resilience in the face of a limited class of attacks. Next, I consider the effect of selfish behavior on the distribution of data and present a practical auditing approach designed to encourage fairness in P2P streaming. I demonstrate the effectiveness of the proposed approaches through experiments using a custom built event-driven simulator and when appropriate, with an implementation of the system evaluated over the Emulab testbed.

I also address the problem of heterogeneous upload bandwidths amongst users, and propose and evaluate a hierarchical approach that allows dissemination of live streaming data in such scenarios. The approach relies on filtering the contents being disseminated, and on providing higher quality data to nodes with higher upload rates. One of the components necessary for the proposed approach is a tool that allows nodes to collect statistics about the overall upload bandwidth distribution across the system. Motivated by this requirement, I finally present NightWatch, a tool that allows nodes to estimate distributions of values held by other nodes (e.g. their upload bandwidth) in a decentralized and inexpensive manner. Such a tool has broader applicability, being also useful for monitoring purposes in large-scale distributed systems.

BIOGRAPHICAL SKETCH

Maya Haridasan was born in Brasília, Brazil on February 5th, 1981, two years after her parents migrated from the southern state of Kerala, in India. During her first year in high school, convinced by her brother, she took part in a computer programming class, which sparkled her interests in Computer Science. Motivated by the field, in 1998 she started her undergraduate studies in Computer Science at the University of Brasilia, Brazil, acquiring the Degree of Bachelors in Science in 2003. Soon after graduation, she moved to Ithaca, NY to further pursue her interests in Computer Science at Cornell University, from where she earned a Masters Degree in 2007, and expects to earn a Doctorate Degree in 2008.

Dedicated to my parents.

ACKNOWLEDGEMENTS

I have thoroughly enjoyed the five years that I have spent at Cornell. I would not have reached this point without the people that surrounded me.

First, I would like to thank my advisor, Robbert van Renesse, for all the teachings, encouragement and patience throughout these years. Robbert had a particular talent for lifting my mood whenever I came into meetings unmotivated about results or anxious about research perspectives. Furthermore, I am thankful for the trust he placed in me, which helped me gain confidence in making decisions by myself. I have been lucky to have such a brilliant, humble and kind advisor.

I would like to thank Ken Birman, for always actively taking interest in my progress. I am grateful for all the compliments and criticisms, which have helped me improve my technical skills, and for the encouragement to keep moving along. I appreciate that Ken has always tried to understand my difficulties and, in subtle ways, encouraged me to overcome them.

I am grateful to Geri Gay, Radu Rugina and Dexter Kozen, for participating in my committee and being supportive of my progress. Thanks as well to all members of the Distributed Systems research group, who have at one point or another, contributed to my experience at Cornell. In particular, to Werner Vogels and Ingrid Jansch-Porto, with whom I had the pleasure of collaborating.

I also cannot forget Becky Stewart, Stephanie Meik, Bill Hogan and Cindy Robinson, who always managed to put a smile on my face as I had to go through the typical paperwork of Graduate School, making everything seem trivial.

Now for some not-entirely-work-related acknowledgements, I would like to thank some friends who contributed to filling these years with fun. Bistra Dilkina, Ellan Fei Spero, Paula Petrica, Filipp Akopyan, Anton Morozov, Basit

Riaz Sheik and Carlos Tadeo Ortega Otero, for the many fun moments, and for proving that different cultures can have fun together. Lori Lorigo and Kiran Gajwani, for showing me that there is actually life during the PhD and that stress is not a mandatory requirement. Anne Marie O'Donnell and Mila Götz, for making our house not only a true home, but undeniably the best place in Ithaca.

Yejin Choi, for treating me with mom-style care when I most needed it. Amrita Basu, for sharing with me the dream of the day when all this would be over and we would take a well-deserved spa trip together. Nosheen Ali and Jui Bhagwat, for sharing so much with me and having gone beyond housemates and friends (now my sisters by choice), and specially for bringing me closer to my roots and feeding my appreciation for *desi* fashion.

I would like to specially thank my family for their eternal encouragement and support. Without them, I would never have aimed so high, and I have no words to thank them for all their dedication. To my brother, who knowing my interests and abilities, encouraged me to study Computer Science in the first place, undeniably one of the best decisions I have made in my life (Who knows what boring job I would now have had I not listened to him?) .To my mother, whose sweetness and wisdom never fail to surprise me. And to my father, who, as much as I hate admitting it, was, is, and will always be right about everything.

And last but not least, I would like to thank Ilya Ganusov, an unexpected and invaluable treasure that I take from this journey through Cornell. For enduring all the yelling and the crying that no one else saw, always providing me with the best and most sound advice, and for making me a more confident and positive person.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

Access to multimedia content on the Internet has grown rapidly in recent years and currently accounts for a large fraction of Internet traffic. This growth has been driven by different styles of multimedia dissemination, such as file sharing, on-demand streaming and live streaming, each of which requires that different timing and delivery requirements and challenges be addressed.

One driver for this trend involves *file sharing*, currently one of the most popular styles of content distribution. In these systems, data is fully available prior to dissemination. The main goal of file sharing systems is that nodes receive the entire data within as little time as possible, without constraints on the order in which blocks arrive. Systems like Napster [42], Gnutella [3], Kazaa [64] and BitTorrent [24] are well known examples of peer-to-peer file sharing solutions; in these systems, client systems collaborate to speed the delivery of desired data.

On-demand streaming is another popular style of distribution. Again, data being disseminated is fully available in advance. With streaming, data should ideally be received at a constant rate. The order in which packets are delivered is important since data is used as it is received. On-demand distribution is suitable for media files that can be reproduced at any time, with loose timing constraints with respect to the delay between user request and start of playback. YouTube [110] is a typical example of an on-demand streaming service, and several media companies such as ABC [1] and NBC [74] are now using on-demand

streaming to disseminate episodes of popular television programs to interested users, among other uses.

The focus of this dissertation is on the *live streaming* style of dissemination, where data should be received by interested users within a short interval from the original transmission by the sender. Live streaming presents a different set of challenges when compared to on-demand streaming, since data is not necessarily available prior to the dissemination and several users are interested in receiving the data almost simultaneously. This style of distribution is useful to broadcast live events in close to real time and to broadcast television over the web. In China, for example, live streaming has become very popular. Applications like PPLive [84, 48] are used to propagate multiple channels simultaneously to hundreds of thousands of users over the web.

The Promise of Internet Television

Live streaming is often associated with *IPTV* and *Internet Television*. The term IPTV typically refers to IP-based services provided by telecommunication companies, which invest heavily on infrastructure to build a proprietary network that can ensure high quality delivery of television content to subscribers, similar to cable television. Despite using IP technology for the delivery of contents, IPTV has little in common with the traditional web philosophy, since it does not give arbitrary users the power to stream media on a global basis. As such, even though IPTV has been claimed to satisfy all needs for web-based streaming services, in reality it does not meet the need for a service potentially accessible to any interested publisher.

While IPTV services provided by telecommunication companies are probably best suited for broadcasting popular channels with high-quality stream rates, streaming over the public Internet allows for a wider range of publishers and contents, favoring diversity and satisfying a wider range of interests. Internet Television refers to this broad concept of sending stream over IP networks, under which the live streaming model studied in this dissertation fits. Internet television refers to a more accessible framework, where the provider of a stream does not control the entire media over which it is disseminated. Streams are disseminated over existing infrastructure, which makes it accessible to any content provider.

Implementing a live streaming system as envisioned by the promise of Internet Television requires protocols that do not impose excessive costs on the publisher. Several protocols have been designed and employed to achieve this goal. Next, a brief description of the current technology behind live streaming is presented. A more detailed presentation of the most important live streaming systems to date is postponed until Chapter 2.

Application-Level Multicast

The trivial approach of sending data from the source of a streaming session to each of the interested clients through unicast connections does not scale to a large number of clients, since the bandwidth cost at the source becomes prohibitively expensive. Several research efforts have attempted to employ more scalable alternative solutions.

Initial approaches attempted to implement multicast distribution at the IP-layer [28, 29]. IP multicast, if widely deployed, would present an efficient and simple solution for implementing live streaming dissemination over the web. Unfortunately, despite earlier promises, IP multicast presents several disadvantages that have prevented it from being widely deployed. It adds complexity at the IP layer by requiring routers to maintain state, and makes it hard for systems to provide higher level features like reliability, congestion control and security.

To overcome this problem, several application level multicast protocols (ALM) that rely on clients' upload resources have been designed and widely studied as an appealing alternative to IP multicast, and previous work has shown that they can be as efficient without incurring large overheads [23, 53, 17, 62, 111, 76, 100]. In ALM protocols, members are organized into an application-layer topology (*overlay*), which is used to define neighbors with which members establish point-to-point connections. With this peer-to-peer (P2P) style of dissemination, where clients contribute with upload resources to the streaming protocol, the promise is that anyone may start their own live streaming session to any number of clients.

In ALM protocols, the sender first breaks down the data being disseminated into packets at the application level. Early ALM systems employed a dissemination tree rooted at the sender, through which all data would flow to all the nodes [23, 53]. In tree-based streaming systems, the sender forwards packets as soon as they are ready to its immediate descendents in the tree, and each node in the tree repeats the process, forwarding the received data to its own descendents.

But these tree-based systems turned out to have a few problems. First, in these approaches, internal nodes in the tree are single points of failure, and if any of them fails, all of the failed nodes' descendents stop receiving data until the tree is repaired. Another problem is that the load of forwarding packets is not fairly distributed across all nodes: leaf nodes do not forward any data.

Given the lack of resilience and fairness of single-tree approaches, tree-based approaches that attempt to equalize the forwarding load across all nodes in the system [17, 62, 100] were employed. These second generation approaches work by constructing multiple paths for the flow of data, and then send different packets down different paths. Even though they enhance fairness in the system, such techniques also require complex protocols for building and maintaining multiple paths.

Attempting to avoid the costs and complexity incurred by tree-based streaming approaches, several *data-driven* (also called *mesh-based* or *pull-based*) approaches have been employed [111, 76]. Nodes are organized into a connected mesh, where each node is assigned a given set of neighbors. In these approaches, packets are not pushed down to nodes. Nodes only push notifications about received packets to their neighbors, and use some scheduling protocol to choose where to request each packet from. The fact that nodes may now fetch packets from different sources makes data-driven protocols more resilient to churn and simple failures. Furthermore, they are simpler to implement and deploy since no complex tree maintenance protocol is required. Nodes instead only need to maintain partial membership knowledge about a few neighbors with which they exchange packets.

1.2 Challenges

Despite significant advances in P2P live streaming functionality, previously employed protocols still present problems that need to be overcome and that hinder the adoption of these technologies. This dissertation focuses on three of these problems and attempts to employ a set of techniques to address them.

First, we consider security vulnerabilities of existing streaming systems and how these can be exploited by malicious attackers. Next, we explore problems caused by the presence of non-cooperative (or *freeloading*) members, which do not have malicious intents but can nevertheless severely degrade dissemination quality by reducing the combined upload bandwidth of the system. Finally, we consider the presence of members with heterogeneous upload bandwidths. This is a common issue, but is not addressed in most existing protocols.

1.2.1 Malicious Behavior

The reliance on users' upload resources for propagating data during a streaming session leaves P2P streaming systems vulnerable to malicious behavior. Users may opt to deviate from the specified streaming protocol with the goal of disrupting the dissemination of certain types of content. Nodes may achieve this goal by simply running a modified protocol so that they are capable of participating in the streaming protocol, perhaps without being detected and ejected.

The degree of disruption that can be caused by individual compromised attackers depends on many factors, such as the protocol being used and the location of malicious nodes in the overlay. These effects can be localized and

minimized if the protocol in use has no single points of failure. On the other hand, vulnerable systems like those based on a single dissemination tree can be crippled if a node high in the tree is compromised. Possible attacks on P2P live streaming systems include:

Membership Attacks: The system may be attacked by compromising the underlying overlay or membership protocol on which it runs. For example, systems that run on top of ring-based overlays are vulnerable to *eclipse attacks* [93], in which an attacker gains control over a large fraction of the neighbors of correct nodes, preventing correct overlay operation. Malicious nodes may also mimic degraded but correct members, or accuse other correct members of having failed.

Forgery Attacks: Attacks may involve fabrication and tampering of data being streamed in the system. Given adequate CPU resources, these attacks can be easily avoided by use of a public key infrastructure. However, at high data rates the cost of signatures can become prohibitive, at least with the most popular authentication protocols.

Denial-of-service (DoS) Attacks: Malicious nodes may overload correct peers with requests for packets or large amounts of duplicate packets, or otherwise disrupt them in ways that compromise their ability to contribute to the streaming session.

Omission Attacks: In applications that require low data delivery latencies, send-omission is an especially serious type of attack. By not forwarding all or part of the packets, a malicious node may disrupt the overall system's timing or availability properties. The main problem with this kind of attack is that a node's guilt cannot be easily proved: after all, the network

itself is quite capable of delaying packets, reordering them, or dropping them.

Collusion Attacks: An attacker may compromise a set of nodes and exploit them to perform a coordinated attack to the system, and may orchestrate the attack to confound whatever defensive mechanisms are built into the dissemination infrastructure.

1.2.2 Freeloading

Users may act selfishly, desiring to save their bandwidth resources for other applications. *Freeloading* (or selfish behavior) may be considered a form of malicious behavior, but the number of freeloaders may be much higher than expected numbers of malicious nodes and therefore the harm inflicted to the system can be much worse. Freeloading has received particular interest and been amply studied in the context of file-sharing systems [3, 56, 33].

In order to avoid freeloading, BitTorrent employs a *tit-for-tat* approach where nodes, when exchanging packets, favor neighbors that are able to provide them with data [24]. The tit-for-tat model has also been explored in BAR Gossip [65] to avoid freeloading behavior in a live streaming system. However, the approach requires the source to stream data to a fixed percentage of the interested users in order for the protocol to work efficiently, which limits the scalability of the approach.

1.2.3 Heterogeneous Bandwidths

The third problem we explore in this dissertation is related to users' available bandwidth, which is essential for dissemination of data in P2P systems. Most P2P live streaming protocols assume that all peers have homogeneous upload bandwidth connections, and that it is enough to upload data at the same rate as the data being disseminated. This assumption is not always realistic. Asymmetric Internet connections are very common and can create conditions under which some nodes have upload rates lower than their download rates. Furthermore, connection bandwidths vary significantly across the Internet. Fairness argues that nodes with higher upload bandwidths should be able to download higher quality data.

Addressing the issue of heterogeneity is complicated by our desire to combine heterogeneous behaviors with fairness requirements (in terms of upload resources spent by nodes). Systems built to overcome freeloading behavior often rely on equal contribution of resources, leading to protocols that only work with nodes of homogeneous bandwidths. In this dissertation we explore this problem and employ an approach which is able to combine heterogeneity and resilience to freeloading into a live streaming system.

1.3 Contributions

To summarize, this dissertation presents techniques that improve the resilience of P2P live streaming systems to malicious attacks and that support bandwidth heterogeneity. First, we present SecureStream, a system composed of techniques

that improve the resilience of live streaming in the face of a limited ratio of malicious attacks. Next, we study the effect of freeloading nodes on live streaming, not previously considered, and employ auditing techniques to enforce fairness in a homogeneous streaming scenario.

We also present an approach that extends pull-based streaming to account for nodes with heterogeneous bandwidths, while using a meaningful definition of the fairness value property. Finally, we present an inexpensive gossip-based approach to estimate distributions in a system, which is a useful tool for nodes to implement the heterogeneous extension just mentioned. A brief description of these contributions is presented below.

SecureStream: An Intrusion-Tolerant Live Streaming System

Our first contribution is the design, implementation and evaluation of *SecureStream*, a P2P live streaming system that limits the opportunity for an attacker to compromise the quality of a streaming session, without incurring a high computational or network overhead [44, 45]. SecureStream combines a set of simple techniques to improve resilience of streaming against the set of attacks presented in Section 1.2, ensuring that malicious attackers cannot gang up against any particular node.

To repel forgery attacks, the system employs an efficient packet authentication technique based on computing and distributing verification digests. To prevent attacks on the overlay structure (the membership protocol on top of which multicast systems operate), SecureStream is built upon *Fireflies*, a scalable one-hop Byzantine membership protocol [55]. *Fireflies* is a probabilistic proto-

col, in which members are presented with a reasonably current view of which members are live or not.

SecureStream uses pull-based packet dissemination. This approach is attractive because it offers participants a choice among multiple candidate packet sources. Because participants are not dependent on any particular peer and can immediately react to failures or attacks, attacks are less damaging. We evaluate the resilience of SecureStream in the presence of varying percentages of attackers and present simulation and emulation results that show that, for up to a limited percentage of attackers, the system is able to sustain satisfactory streaming quality.

Audit-Based Approach for Improving Fairness

After evaluating the potential harm inflicted by freeloading behavior on live streaming protocols, we employ auditing techniques to encourage data-sharing [43]. Our auditing approach establishes a minimum threshold for the amount of data sent by any node in the system, and removes nodes that upload less data than the threshold. Instead of relying on a tit-for-tat mechanism, we focus on encouraging nodes to respect the established protocol. Nodes are forced to provide accountable information regarding packets sent to and received from neighbors, and the auditing system is responsible for detecting and removing misbehaving nodes.

Identifying misbehaving nodes is not a trivial task, since there is no fixed minimum amount of data that each node must contribute to the system. If we assume a model where misbehaving nodes simply do not upload any data, de-

tecting them would be an easier task. However, once we assume that misbehaving nodes may adjust their contribution level based on the policy used by an auditing system, a more elaborate approach is required. We present and evaluate an auditing model based on sampling the system and using the sampled information to build a global view of how the system is currently behaving. Based on it, auditors employ strategies to identify the misbehaving nodes that should be punished.

Heterogeneity-Aware Live Streaming

We looked into the problem of bandwidth heterogeneity among nodes in a streaming system and employ an approach that allows taking heterogeneity into consideration. We argue for fairness in the dissemination process, by which we imply that nodes that contribute with more data should also receive higher quality data. Meanwhile, nodes that are incapable of contributing, or unwilling to contribute enough upload bandwidth, receive data at the best possible rate allowed by the outstanding upload resources.

In this work we explored node placement as well as multi-layered video coding techniques to achieve a scalable and adaptive substrate for streaming to nodes with heterogeneous upload bandwidths. Different node placement techniques were explored and analyzed based on metrics such as fairness and quality of data delivery. Our approach organizes nodes into hierarchical layers according to their upload capacities: higher-level layers receive and propagate better quality data to other members of its layer and filter data that is propagated to members of lower-level layers.

NightWatch: Gossip-Based Distribution Estimation

In live streaming systems where nodes have heterogeneous bandwidths, it is useful to rank nodes based on their upload bandwidths, in order to allow nodes to implement a decentralized algorithm for organizing themselves into hierarchical layers. This dissertation's final contribution is the description of NightWatch, a tool that allows nodes to estimate the distribution of values held by other nodes for particular properties (for example, upload rates) [46].

Such a tool has broader applicability, as it can help many kinds of systems achieve enhanced resilience and sophisticated forms of self-adaptation. We designed an approach where nodes build estimates of the distribution of values, obtaining results in a timely and scalable manner. Our approach relies on gossip-based exchange of data, and uses data synopsis techniques for minimizing the amount of data exchanged between pairs of nodes. Nodes maintain a fixed-size array of entries and periodically exchange and accumulate information obtained from other peers.

We evaluated our gossip-based approach through simulation, employing four candidate synopsis techniques. We compared these in terms of quality of the estimation, storage and bandwidth requirements, and convergence time. All techniques were evaluated with a diverse set of distributions, including uniform, normal, heavy-tailed, and bimodal distributions. By experimenting with up to a hundred thousand nodes, we empirically validated that a limited number of rounds and a constant message throughput per node in each round is sufficient to achieve an efficient and lightweight protocol.

1.4 Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents related work on P2P live streaming and vulnerabilities of P2P dissemination systems. Chapter 3 studies vulnerabilities of mesh-based live streaming systems and presents SecureStream, a system that combines techniques that may be used to build a system resilient to certain classes of malicious attacks. Chapter 4 looks into the problem of freeloading behavior, and employs a simple audit-based approach to ensure the quality of the dissemination process. Both chapters 3 and 4 consider systems where all nodes have homogeneous network bandwidths. Chapter 5 investigates how to build a live streaming system capable of handling nodes with heterogeneous bandwidths. Chapter 6 presents NightWatch, a gossip-based tool for estimating distributions of values held by nodes in a distributed system. Chapter 7 considers future work and summarizes the presented work.

CHAPTER 2

RELATED WORK

This chapter provides an overview of the problem domain explored in this dissertation with a focus on work related to ours. First, we review background work on live streaming systems. We describe the most important protocols previously employed and that have led to the current state of art, exposing the evolution of the paradigms underlying these protocols. As part of this overview, we also describe related work on improving the resilience of live streaming systems to unexpected behavior of internal peers. In the second section, we shift the focus from live streaming systems and present work related to our NightWatch system, providing an overview of previously explored gossip-based techniques that estimate aggregates of information in distributed systems, for monitoring or other purposes.

2.1 Live-Streaming Dissemination

We start by describing the most relevant streaming protocols employed to date, to the best of our knowledge. It should be noted that there are a huge number of such protocols, and we will not attempt to be exhaustive. We group existing protocols into two main groups, based on the level at which they are implemented, and into subgroups based on their style of packet dissemination.

The first efforts to implement dissemination systems argued for implementation at the Internet IP level. Later, tree-based protocols in which routing is performed at the application layer on end hosts were employed, followed by data-driven protocols. Finally, more recent protocols have attempted to com-

bine and further improve previous approaches, and improve the resilience of the same.

2.1.1 IP Multicast

The standard multicast model for IP networks was first proposed in the late 1980s [28]. According to the original model, IP multicast was designed to use the UDP protocol to provide unreliable delivery of packets to dynamic groups, which members can join and leave when desired. Furthermore, the source of a multicast does not need to have membership information about the multicast group: it suffices to know the address associated with the group.

IP multicast attempts to deliver streams of information to multiple receivers without burdening the source or the receivers, while still reducing traffic on the network. Figure 2.1 illustrates IP multicast routing from a streaming server to a multicast group containing interested clients. Routing of packets from the source to members of a group is performed by routers at the IP layer, who store information regarding group membership. In order to specify how far to forward packets, IP multicast routing protocols use the Time To Live (TTL) field of IP datagrams. The default TTL value of 1 results in multicast packets going only to other hosts on the local network. Higher TTL values increase the maximum number of hops that routers will forward a packet.

A special class of IP addresses (class D) is reserved for IP multicast groups. Nodes are free to join or leave any group at any time. Nodes express interest in receiving packets by subscribing to a particular group, which is performed by sending a message to the nearest router following a *Group Management Pro-*

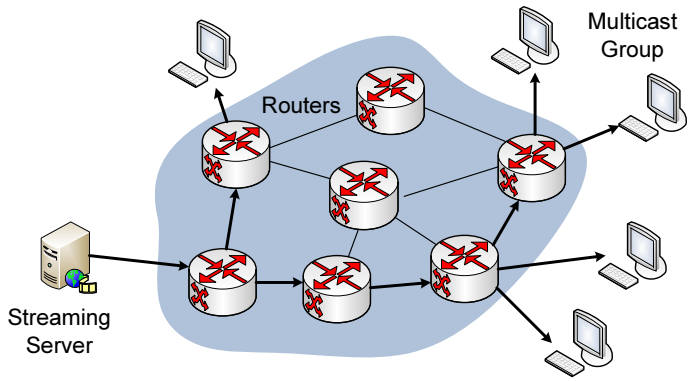


Figure 2.1: IP Multicast Routing

tocol. Such protocols are employed by routers to learn about the presence of group members on their attached sub-networks. The Internet Group Management Protocol (IGMP) [16] and the Multicast Listener Discovery (MLD) protocol [30] are the defined Group Management Protocols for IPv4 and IPv6 systems. Systems that rely on non-standard group management protocols have also been employed (e.g. [49, 26]).

IP Multicast requires routers that can support it. Since not all Internet routers supported IP Multicast [29], initial research relied on a virtual network built on top of the Internet [32]. The *Multicast Backbone* (MBone) shares the same physical media as the Internet while using a parallel system of routers that can support multicast. MBone is composed of networks that support multicast, each of which runs a multicast routing demon. These deamons are connected with one another via unicast tunnels. Later, two Internet2-based backbones, namely *very-high-speed Network Backbone Network Service* (vBNS) [52] and *Abilene*, besides the commodity Internet, have also been used to study and deploy *inter*domain IP Multicast protocols.

Intradomain Protocols

Initial IP multicast efforts were focused on intradomain protocols. In these protocols, hosts interested in transmitting data to a multicast group send data to their nearest router, which takes responsibility for forwarding the data to other routers involved in the multicast for the particular group. Data is thus forwarded by routers towards interested nodes, following a tree path through routers. The path followed by packets is defined by the routing protocol. Some multicast routing protocols build a single tree path for each dissemination source (*source tree approach*), while others build a tree which is shared by all sources (*shared tree approach*).

Source tree protocols build a single shortest path tree (SPT) for each multicast group, which is rooted at the source of the multicast stream. Building the tree for a group is usually performed using Reverse Path Forwarding (RPF) [27]. Such protocols have the advantage of providing the most efficient path to receivers of each group, but require routers to keep a large amount of information, given that each source requires a specific dissemination tree.

The Distance Vector Multicast Routing Protocol (DVMRP) was the first source tree protocol used to make routing decisions on the MBone [104]. It creates trees using a broadcast-and-prune technique, by which packets sent by the source are forwarded downwards by routers, until prune messages are received from routers downstream. Prune messages are sent back by a router towards the source to indicate that its downstream interfaces do not lead to any members of the desired group. Protocols that follow this broadcast-and-prune technique are also known as dense mode protocols, in reference to their adequacy for densely populated topologies. Other source tree protocols include the Multicast Exten-

sions to OSPF protocol (MOSPF) [72] and the Protocol Independent Multicast (PIM-DM) [2].

Building individual trees for each multicast source incurs significant complexity and space overhead at core and edge routers. In order to avoid these costs, shared tree protocols have been employed. Shared trees use a single common root placed at some chosen router in the core of the network, called the *rendezvous point* (RP), to bring sources and receivers together. These protocols require significantly less storage at the routers at the cost of introducing some latency in packet delivery since paths are no longer optimal. Core Based Tree (CBT) is an example of a shared tree protocol [11].

Adaptive solutions have also been proposed. The Protocol Independent Multicast - Sparse Mode (PIM-SM) approach combines the advantages of both shared and source-based trees [34]. The protocol switches between a shared tree and a shortest path tree if performance thresholds are not satisfied, attempting to find the right tradeoff between complexity at the routers and packet latency depending on the multicast scenario.

Interdomain Protocols

All these protocols were designed without much consideration to scalability, and are only applicable in intra-domain settings. Some solutions have been developed to support inter-domain routing, such as the Multicast Border Gateway Protocol (MBGP) [97] and the Multicast Source Discovery Protocol (MSDP) [35]. The combination of MBGP, PIM-SM and MSDP aims to make multicast routing hierarchical, extending previous approaches.

MBGP is an extension to the BGP protocol [89] used by border routers to reliably exchange network reachability information. MBGP's main role is to provide next-hop information between domains, which routing protocols use to find the best path towards sources. MSDP further helps achieve interdomain multicast by allowing a rendezvous point in one domain to find out about sources in other domains. The protocol works by having representatives in each domain announce the existence of sources to representatives in other domains.

The MBGP/PIM-SM/MSDP combination has been considered only a short-term solution because of scalability concerns [7]. Several other approaches have been designed aiming to provide more elegant solutions to the problem of intra and interdomain routing [96, 63, 10, 49, 57, 86].

Problems with IP Multicast

Despite the advances in IP multicast technology and its increase in use within commercial settings, a few issues have prevented it from being deployed at a global scale. One of the main problems with IP Multicast is the complexity and space overhead that it requires at core and edge routers, violating the popular end-to-end argument [91], which argues that whenever possible, intelligence should be pushed to the endpoints of the network [108]. IP-multicast routing protocols require routers to maintain large routing tables, and impose high computational and network overheads.

Router migration is another important consideration, since older hardware do not support IP Multicast. Furthermore, most solutions lack simple and scalable mechanisms for supporting access control, protection against routing at-

tacks, address allocation and network management [31]. Firewalls and lack of support for Network Address Translation (NAT) further hinder global adoption of IP Multicast. These problems have led to a slow adoption of IP Multicast on large scale settings in the Internet.

2.1.2 Application-Level Multicast

In response to IP multicast's slow deployment in global settings, research focus on live streaming moved from the network layer to the application layer, starting with the End System Multicast (ESM) service [23]. This and other *Application-Level Multicast (ALM)* systems implement all multicast-related features at the end systems. In ALM systems, nodes are organized into an *overlay*, which denotes a logical network running on top of the Internet. Communication between nodes in the overlay relies on IP unicast and therefore does not require changes to the underlying infrastructure.

Single Tree

Some of the first ALM protocols relied on the use of dedicated servers for the dissemination of data (Scattercast [21] and Overcast [53]). Other protocols, like Yoid [36] and ALMI [78], were the first ALM protocols to argue for an entirely peer-to-peer architecture. In this later architectural model, nodes participating in the stream not only receive but may also upload data to other nodes, and little or no extra dedicated infrastructure is necessary. Independent of the basic architecture, the first generation of ALM systems relied on approaches based

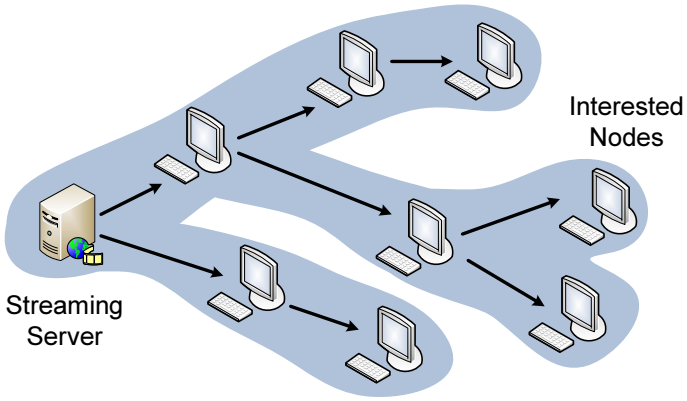


Figure 2.2: Single-tree application-level multicast

on pushing data down to interested nodes through a single dissemination tree rooted at the source of the stream (Figure 2.2).

Narada [23], which is the protocol used for routing packets in ESM, was one of the first tree-based streaming ALM systems. In Narada, nodes interested in the stream are organized into a mesh structure using a decentralized protocol. The mesh is constantly updated by nodes, which periodically search for better neighbors, adding and removing links based on some utility function. Routing algorithms are then used to build dissemination trees rooted at the source, using only connections from the underlying mesh.

Scattercast [20, 21] employs a combination of dedicated proxies, which are connected to each other via unicast connections, and locally-scoped multicast groups. Interested nodes either connect to the closest proxy through a local IP multicast group, or (if this is impossible) create a unicast connection. Similar to the mesh maintained by end systems in Narada, proxies in Scattercast are organized into a dynamic mesh and periodically run a local mesh optimization protocol. Dissemination trees are built for each source and multicast group.

In Overcast [53], nodes are organized into a dissemination tree that adapts to the underlying network conditions. The tree is constructed with the goal of optimizing the download bandwidth of users. Nodes initially join the tree by contacting the source, and then recursively compare the bandwidth achieved with its current parent to the bandwidth it would achieve if it connected to any of the parent's children. Furthermore, nodes periodically check if they can improve their bandwidth by testing connections with their siblings, parent and grandparent.

Rendezvous points (RPs) are used by Your Own Internet Distribution (Yoid) [36] and by ALMI (Application Level Multicast Infrastructure) [78]. Yoid is a peer-to-peer protocol which maintains one or more RPs associated to each multicast group, which are dedicated nodes. These RPs maintain a list of current group members, querying them periodically to check for their liveness, and serve as a bootstrap mechanism. Members contact a group's RP whenever they are interested in receiving data from that particular group. Meanwhile, the Yoid Topology Management Protocol (YTMP) organizes nodes into a tree topology for dissemination. At first, nodes request from a RP a set of members of a multicast group, and connect to the first member whose maximum set of children has not yet been reached. Later, nodes periodically test connections to other members discovered through the RP in search for lower latencies.

ALMI [78] was designed for large numbers of groups containing a small number of members. Every multicast group has a session controller, which is responsible for calculating the minimum spanning tree overlay for the session members (interested nodes). The controller ensures that the tree is always connected, providing parents for joining nodes and handling node departures. Fur-

thermore, it instructs each member to monitor and collect measurements from a few others, which it then uses to periodically recalculate the minimum spanning tree. The reliance on a centralized approach improves reliability, but does not allow ALMI to scale to large numbers of members within a group.

Some tree-based protocols are built on top of Distributed Hash Tables (DHTs) that provide a routing infrastructure between nodes, such as Bayeux [113](built on top of Tapestry [112]), Scribe [18] (built on top of Pastry [90]), and an ALM protocol [88] built over CAN [87]. DHTs are decentralized distributed systems that provide a lookup service similar to a hash table, allowing any participating node to efficiently retrieve the value associated with a given identifier. The best known DHTs use P2P architectures, and achieve logarithmic routing time for packets.The use of DHTs provides multicast protocols with a scalable and often fault-tolerant routing mechanism which can be used to connect nodes and route messages.

Several other systems based on single-tree dissemination of data have been designed [12, 92, 98, 15]. One of the main disadvantages of peer-to-peer protocols based on pushing data through a single tree is that the task of uploading data is not fairly distributed between nodes in the system. In a dissemination tree, only internal nodes propagate data, while leaf nodes simply receive data, not contributing to the dissemination process. Furthermore, such protocols present single points of failure, since the crash of one internal node leads to all its descendants not receiving any data until the tree is repaired.

Splitting the Flow of Data

Later work on peer-to-peer streaming systems focused on improving fairness among peers and resilience to churn (nodes joining and leaving the system). Bullet [62] and SplitStream [17] were two of the first protocols to address the lack of fairness of previous protocols. Later, ChunkySpread [100] enhanced the benefits provided by splitting the load more evenly.

Bullet distributes the load among nodes by breaking the data stream into packets and sending them to peers through disjoint dissemination paths. The protocol achieves this behavior by combining mechanisms from both tree and mesh-based overlays. Packets are pushed down through different paths in a constructed overlay tree and are then exchanged between peers through random perpendicular links along the tree. Bullet relies on existing protocols for building the initial distribution tree. The source of the tree chooses a subset of its children to send each packet to, which recursively repeat the process and randomly choose subsets of their children to propagate the data. An important component of Bullet is the RanSub protocol, which is used to disseminate information about which packets each node holds, so that nodes can later use perpendicular links to obtain missing packets from peers that have received them.

SplitStream [17] is a multiple-tree dissemination protocol, which splits the streamed data into stripes and disseminates each stripe through a different dissemination tree. The trees used for dissemination consist of disjoint sets of internal nodes, leading every node in the system to be an internal node in at most one tree. This property leads to an ideally fair system, where every node uploads data at the same rate as the stream being disseminated. SplitStream uses

the Scribe multicast protocol [18], which in turn uses the Pastry DHT [90] for building dissemination trees. Scribe first identifies a rendezvous point (RP) by requesting that a Pastry node route a packet using the group identifier as the key. The node reached is defined as the RP of the group, and the dissemination tree is built by combining the routing paths from the RP to each member of the group. SplitStream uses separate Scribe multicast trees for each of its k stripes. Despite improving the load distribution across all nodes, SplitStream is a complex protocol and does not avoid tree-related vulnerabilities to failures.

ChunkySpread [100] uses a protocol reminiscent of SplitStream. It breaks data into k stripes and uses multiple dissemination trees. Unlike SplitStream, ChunkySpread allows nodes to contribute with heterogeneous upload bandwidths to the system, based on their capacity and willingness. In ChunkySpread, nodes are organized into an overlay mesh, where the number of neighbors a node has is proportional to its available upload capacity. By periodically exchanging local information with neighbors, nodes select parents and children to maximize the stream quality taking load and latency into account.

One of the main criticisms to most of these protocols is the complexity and costs involved in building and maintaining the required dissemination trees in the face of churn. With the exception of Bullet, all presented protocols are purely push-based protocols, which do not allow for easy recovery from lost data transmissions.

Pull-Based Systems

More recent work on peer-to-peer systems has focused on a different paradigm for the dissemination of data. Instead of the traditional model of pushing data towards all nodes, in these new protocols nodes are organized into an overlay mesh and request data from other peers based on knowledge of what packets each peer holds. This model is commonly called *data-driven*, *pull-based* or *mesh-based* streaming, all of which are indicative of properties of these systems. Mechanisms for exchanging information on what packets each node holds and for requesting packets are the focus of protocols like CoolStreaming [111] and Chainsaw [76], which were two of the first pull-based protocols.

Coolstreaming presents a simple data-driven approach [111]. In Coolstreaming, nodes periodically exchange information about data they hold with a set of partners, and request missing data from these nodes based on the collected information. Nodes join a session by contacting the source node, which provides it with a deputy node that will then provide it with a list of candidate partners. Nodes dynamically update their set of partners through use of the Scalable Membership Protocol (SCAMP)[38], which allows distribution of membership messages among participating nodes.

Each node in CoolStreaming maintains a *Buffer Map* (BM) indicating the set of packets it currently holds, and periodically exchanges its BM with its partners. Based on the collected maps, a node uses a scheduling policy to fetch missing packets from its partners. In their heuristic algorithm for scheduling, nodes first calculate the set of potential suppliers for each missing packet. Packets with fewer suppliers are scheduled first, and suppliers with highest available bandwidth are chosen.

Chainsaw also relies on a membership protocol for organizing nodes into a mesh overlay, but is not tied to any particular protocol. In Chainsaw, nodes use a simple policy for requesting packets from neighbors, randomly fetching packets from those with available packets, only respecting a limit on the number of outstanding requests. Nodes use notifications to inform neighbors of the receipt of new packets, instead of the buffer map used by CoolStreaming. Since nodes are informed of the receipt of packets immediately after their neighbors receive packets, Chainsaw presents smaller latencies for the receipt of packets compared to the Coolstreaming protocol. Our work is heavily influenced by the Chainsaw protocol, and therefore, a more detailed description of the protocol is presented in Chapter 3.

A similar pull-based approach is used in PRIME (Peer-to-Peer Receiver-Driven Mesh-Based Streaming) [68]. The main difference between PRIME and Coolstreaming or Chainsaw is that nodes are connected through a randomly connected but directed mesh, providing each node with multiple parents and multiple children. Much as was the case in Chainsaw, notification of new packets are pushed from parents to children, so that nodes can request packets from candidate parents. Comparing PRIME and a SplitStream-like protocol, in [69] the authors argue that mesh-based approaches present better performance over tree-based approaches due to their flexibility in the packet distribution process.

Meanwhile, hybrid approaches that combine both push and pull style of data exchange have also been employed. mTreebone [105] uses a set of stable nodes to construct a tree-based backbone through which most data would be pushed, where stability is defined by nodes' ages in the streaming session. In parallel and for repair purposes, all nodes are organized into a mesh overlay,

which complements the backbone and through which data is pulled amongst nodes only in case of data outages. Even though more resilient to churn, the approach presents similar problems to single-tree approaches in terms of fairness among nodes in the uploading process.

2.1.3 Improving Reliability

Peer-to-peer systems are vulnerable to rational and malicious behavior because nodes providing service are untrusted, and may deviate from the intended original behavior. In this subsection we present work that addresses the vulnerabilities of previous approaches, improving reliability of live streaming systems in the face of misbehaving peers in the system.

Ngan et al. [75] consider fairness issues in the context of tree-based peer-to-peer streaming protocols. The authors present mechanisms that rank peers according to their level of cooperation with the system. One of their techniques involves the reconstruction of trees as a way of punishing opportunistic nodes. Most of their mechanisms require peers to keep track of their parents' and children's behavior.

PULSE [81] is a P2P live streaming system that tries to reward nodes that contribute resources and discourage peers from contributing an insufficient amount of resources. The main idea consists of using a pull-based dissemination protocol and moving nodes that contribute with more upload resources closer to the source, leading to lower packet latencies. The system was evaluated in heterogeneous settings, showing that nodes with higher upload capacity have a smaller latency compared to less favored nodes.

Oversight [25] is a framework that enforces download rate limitations on P2P media streaming systems. The protocol relies on a set of trusted nodes that store information on the data downloaded by each node receiving data. Nodes only send an object after consulting the trusted nodes to verify if the nodes requesting the stream are not over-requesting data. Oversight is targeted at systems where nodes upload entire media objects from each other, and not for live streaming systems where all nodes are interested in receiving the exact same data in close to real time.

Pai et al. studied the effect of different types of incentives on the Chainsaw protocol [77]. After exploring tit-for-tat and some variations, the authors present an algorithm that sets up local markets at every node, where neighbors compete for the node's upload capacity. Nodes favor neighbors that contribute more. Experiments were limited, with nodes classified as fast or slow nodes. The results indicate that the proposed algorithm improves the performance of the system when the total upload capacity is not enough to supply all the nodes.

Drum [9] targets DoS attacks on gossip-based multicast protocols, eliminating vulnerabilities to such attacks. The main idea in Drum is to have half of the links of each node be picked by the node itself, and half be picked by other peers. That way, even if only malicious peers contact a node, the peer can still get correct data from the peers that it picks. The authors showed that the approach works well for multicast protocols which do not have time delays, but have not studied its performance for multicast systems where a high throughput of packets is desired and the upload capacities are limited.

BAR Gossip [65] is a live streaming approach that tolerates the existence of selfish and malicious nodes. Time is divided into rounds, in which each peer

communicates with another peer selected using a pseudo-random function. In each round, peers exchange their current history containing the identifiers of all the current data, as basis for the next exchanges. Nodes also perform a phase of *optimistic push*, forwarding useful updates to another pseudo-randomly picked peer with no guarantee of useful return. The approach requires that the broadcasting source have full knowledge of all members in the system and always unicast each update to 5% of the nodes, a limitation on scalability. Despite its limitations, BAR Gossip was the first streaming system to successfully employ bartering to enforce fair contribution of nodes.

A general technique to improve fault-tolerance in P2P systems based on group-oriented monitoring is proposed in [37], which could be applied to small scale live streaming. An abstraction called *link attestation groups* is employed to allow nodes to monitor one another within a group. As long as correctness properties are defined by developers, nodes may verify their neighbors' behavior and submit digitally-signed attestations. These attestations are shared among group members and used by nodes to build link attestation graphs. Our approach to enforce fairness in live streaming pull-based systems shares some ideas with this work, namely that nodes monitor one another to identify incorrect behavior.

2.1.4 Popular Commercial Systems

Most of the protocols previous described have not been deployed in large scale settings with real users. Coolstreaming [111] is one of the few exceptions, having been employed to propagate data to up to 30000 users with more than 4000

simultaneously being online. More popular P2P systems have been used both for file-sharing (e.g. BitTorrent [24]) and for live streaming (e.g. PPLive [84]). We now briefly review these two systems, given that they have been used in very large scale settings, having thus influenced the design of many protocols later employed.

BitTorrent

BitTorrent [24] is a file-sharing P2P protocol designed and deployed in 2001. Since then BitTorrent has become widely popular, being used by millions of users to download files of interest. Even though it does not present the same challenges as live streaming systems, BitTorrent's role as one of the killer P2P applications has influenced commercial live-streaming systems to follow several of its design ideas.

Each user may share or download files, and to do so, it uses a BitTorrent client, which will exchange packets on its behalf. To each file that is shared, a *Torrent* file is associated, which aggregates metadata about the chunks that compose the file and about the server coordinating the distribution of the particular file (known as the *tracker*). The tracker only manages connections, not keeping any copies of the contents of the files being distributed, and therefore requiring limited bandwidth.

When a peer is interested in downloading a file, it first obtains a torrent file for it, and based on it contacts the tracker for the file (Figure 2.3). The tracker then provides the node with a list of currently active peers who are downloading the same packet or who have finished downloading it and are now acting as

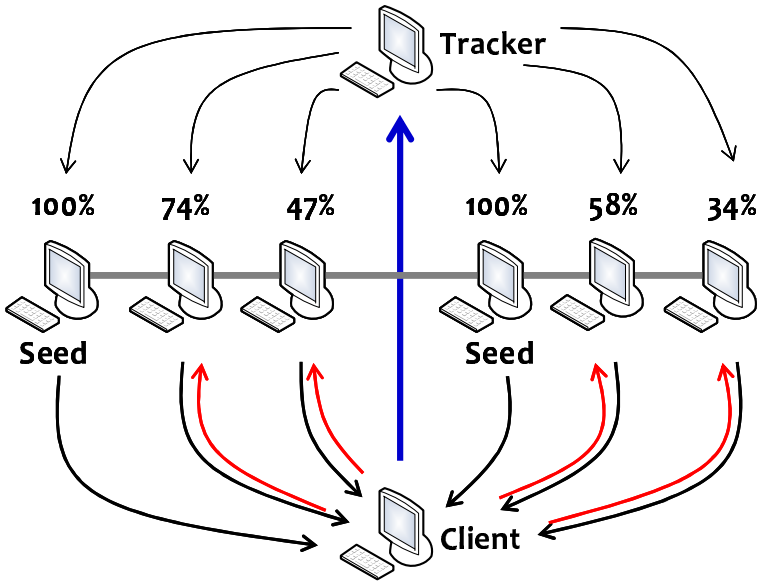


Figure 2.3: BitTorrent Protocol

seeds for that torrent. A set of peers that shares a torrent is called a *swarm*, and initially consists of only one seed.

The interested peer then directly contacts the peers in the received list and starts simultaneously exchanging packets with several of them in arbitrary order (non-contiguous). After an initial bootstrapping phase where nodes only download chunks, they are also able to upload them to other peers, or even to later act as seeds. Notice that the more popular the file being downloaded is, the more peers will be participating in the exchange of packets, therefore leading to faster and more reliable download rates.

BitTorrent clients may employ any mechanisms to optimize their download rates. In order to discourage packet freeloading (by nodes that attempt to download files without contributing to the uploading process), most BitTorrent clients incorporate a bartering approach (also known as *tit-for-tat*) to the exchange of packets between peers: when satisfying requests of packets, nodes favor peers

who also send them back data. To allow newly joined nodes to bootstrap, and to initiate exchanges between each pair of nodes, clients employ a mechanism called *optimistic unchoking*: a portion of each peer's upload bandwidth is reserved for sending chunks to random peers.

PPLive

PPLive is a proprietary system used for live streaming, which has been used to propagate over 400 channels to 400000 users per day (on average), according to its website [84]. Channels' bit rates vary from 250Kbps to 800 Kbps. Despite the fact that it is proprietary, some characteristics of PPLive have been identified and exposed in previous work [103, 48]. Other systems similar to PPLive have also become popular, such as PPStream [85] and SopCast [95].

Each channel in PPLive streams live contents or previously recorded programs through its own overlay of cooperative peers. As in previously described systems, contents are broken into packets and nodes help propagate packets among each other. When a node is interested in receiving a particular channel, besides participating in the overlay for the desired channel, it may also help propagate packets in other channels. Unfortunately, the exact protocol used by PPLive to assign uploading load to nodes is not yet well known.

When joining a channel, a node first retrieves the list of channels from dedicated servers that maintain lists of channels. Later, nodes contact dedicated membership servers to request a set of peers participating in the overlay related to the channel they are interested in. Starting from the received list, each node looks for neighbors, periodically updating its set of neighbors as required (Fig-

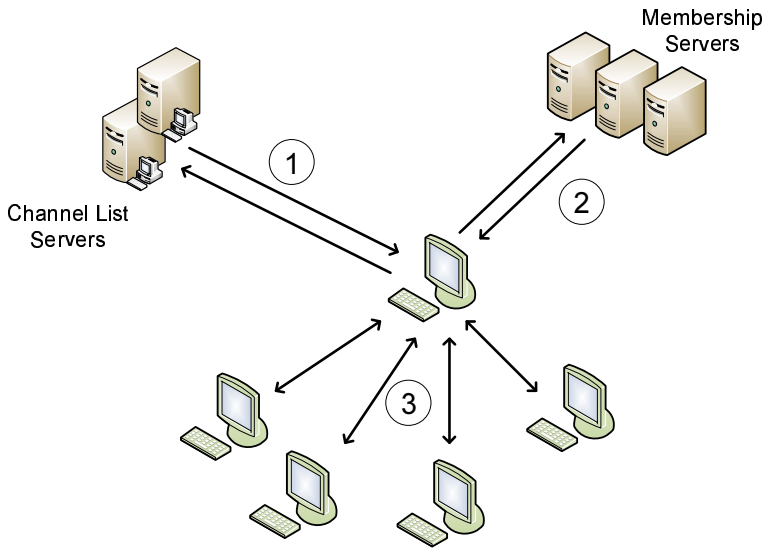


Figure 2.4: PPLive Protocol

ure 2.4). At any given time, nodes always maintain a list of *real* peers, with whom they exchange data, and *candidate* peers, for use in case some of the real peers become unresponsive. UDP packets are used for control packets related to discovery of peers, while TCP connections are used among peers who exchange actual streaming data.

A previous study [103] has shown that the average node degree in PPLive is independent of the channel size, and that the structure of PPLive overlays is closer to that of random graphs, unlike in file-sharing systems. Randomness, however, depends on the channel size, being stronger in smaller overlays. As overlays increase in size, so does clustering of nodes, even though some randomness can still be observed.

2.2 Computation of Aggregate Information in Large-Scale Systems

Our NightWatch system, presented in Chapter 6, employs a gossip-based approach for estimating the distribution of values held by nodes in a distributed system. In this section, we present a brief overview of the work most closely related to NightWatch. First, we present similar work proposing techniques to compute individual aggregate functions of distributed values. Unlike these systems, NightWatch attempts to estimate the distribution of values, rather than individual aggregate functions over the same. Second and last, we present previous work on synopsis construction techniques. A synopsis is a concise representation of a data set, and as such, can provide useful information about general properties of the data. NightWatch relies on the construction of such synopses to concisely represent a large number of distributed values.

2.2.1 Aggregation

The idea of computing aggregate functions in large-scale distributed systems has been extensively studied [13, 99, 109, 54, 5, 73, 67]. The main goal of these approaches is to minimize the amount of data exchanged and the convergence times for computing aggregate values, while providing either perfect or satisfactory accuracy.

In Astrolabe [99], aggregation is employed as a summarization mechanism. Nodes are divided into non-overlapping zones, which are arranged hierarchically. Each node runs an Astrolabe agent, which periodically gossips state infor-

mation with other agents, both within its zone and from other zones. Depending on the zone of the agent with which they exchange information (same zone or not), they exchange information about their own zone or about other zones that they know of. Leaders in each zone collect information about nodes in its zone and compute aggregate values. Since all nodes are involved in each query that an application might be interested in, Astrolabe does not scale well if the number of queries increases significantly.

SDIMS [109] is a management system similar to Astrolabe. SDIMS hierarchically aggregates information about large-scale networked systems and provides a database abstraction for querying the state of the system. The system leverages DHTs to implement a scalable management system. Instead of using a single tree, multiple trees are used to aggregate different attributes, distributing the load among all nodes. The routing infrastructure provided by DHTs is used to determine the nodes that participate in the aggregation tree for any given attribute.

In [13], Bawa et al. introduce an approach that allows any node to issue a query for computing an aggregate function (minimum, maximum, count, sum, average, etc) over distributed data. The paper employs a tree-based solution, with focus on queries issued by a single peer. In its basic scheme, the querying peer broadcasts the query to the network, and a spanning tree is constructed during the dissemination of the message. In the second phase of the protocol, the answer to the query is computed in a bottom-up fashion, with nodes sending the combined results of their children up towards the querying node. Unlike NightWatch, in the proposed model the peer issuing the query is the only one that obtains the information at the end of the aggregation process.

In the Newscast protocol [54], nodes exchange vectors (cache entries) containing several values for computing aggregate values, an idea similar to the one employed in the NightWatch protocol. In each round, peers randomly select another peer to exchange all cache entries they currently hold. The choice of which cache entries are kept after the new entries are received is based on the age of the entries. Only the youngest entries are maintained, and the set of peers associated with each entry constitute the set of neighbors known by the owner of the cache at a given time. The approach was explored for computing extreme (minimum and maximum) mean values.

Work on gossip-based aggregation has also been done in the context of sensor networks, where energy and constant loss of communication are important factors to be considered. TAG (a Tiny Aggregation Service for Ad Hoc Sensor Networks) [67] builds a tree topology to compute aggregates without spending much energy, and avoiding duplicate information. Sensors route data towards the interested querying user through a dissemination tree, aggregating the data on the way to the root. [73] employs the diffusion of synopses, but focuses on finding solutions that avoid double-counting values. By employing techniques that are duplicate-insensitive, different topologies can be used for collecting information, and redundant paths can be explored to avoid loss of data when nodes fail.

The main difference between NightWatch and previous approaches, is that NightWatch aims at estimating the distribution of values spread over a distributed system, rather than specific aggregate functions. Furthermore, unlike most systems NightWatch aims to provide every node with satisfactory estimates, rather than one querying node.

2.2.2 Synopsis Construction

NightWatch relies on the construction of concise synopses of data to minimize peers' storage and bandwidth requirements. The need for efficient synopsis-construction techniques in the context of database systems has led to the proposal of a variety of techniques. Even though the general problem specification is different, previous techniques, if modified, can be employed in the context of estimating distributions of values in P2P systems.

Sampling constitutes one form of maintaining a concise representation of a large set of data. Dynamic approaches based on sampling and storing a fixed number of values aiming to obtain an unbiased sample, are known as reservoir based methods [101, 8, 4]. Values are entered and removed from the sampled set dynamically, as data is processed. To avoid storing duplicate values in the sample, in *concise sampling* values in the sample have an associated counter [40]. Advantages of sampling include simplicity, efficiency, and ease of use, being the only appropriate approach for high dimensional applications.

Several approaches based on the construction of histograms have also been explored [61, 82, 51, 83, 41]. Histograms group values into bins (buckets), which makes them good candidates for approximating the frequency distribution of a large set of values. The use of histograms is straightforward when considering a static set of values, since they can be pre-computed taking all values into account. Histogram-based techniques present two main problems, namely the choice of bucket sizes, and handling dynamic sets of values.

Other more elaborate types of synopsis-based techniques have also been employed, including wavelets [58, 19, 102] and sketches [50], among others.

Wavelets and sketches are well studied techniques for decomposing and summarizing data hierarchically. In the wavelet technique, data is decomposed into a set of wavelet functions and base functions, where the higher coefficients of the decomposition capture the broader trends in the data and the lower coefficients capture the more specialized trends. Sketches can be considered randomized versions of wavelet-based approaches. A thorough survey of synopsis-construction techniques for data streams is presented in [5].

CHAPTER 3

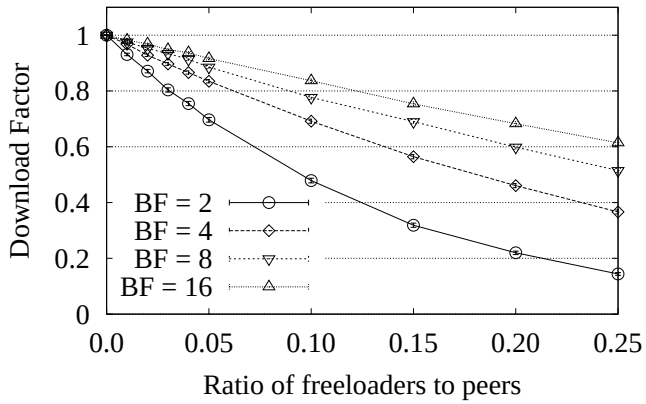
SECURESTREAM: INTRUSION-TOLERANT LIVE STREAMING

3.1 Introduction

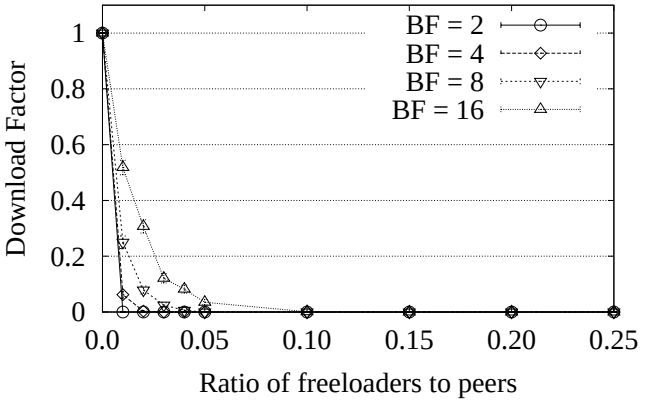
Malicious behavior has long been a serious cause of concern in systems deployed over the Internet. Centralized systems are extremely vulnerable due to the presence of unique points of failure. In contrast, peer-to-peer (P2P) systems present better tolerance to attacks due to their redundant nature, with multiple peers collaborating to the execution of tasks. However, many P2P systems are not entirely decentralized and present unique points of failure, and are therefore still vulnerable to attacks. Moreover, attackers may have control over internal nodes in the system and may explore omission attacks in which compromised peers do not perform their duties, such as forwarding packets to other peers. This type of attack is not trivial to handle since attackers cannot easily be identified and proven guilty, and damage to individual nodes or to the overall performance of the system can be severe.

To illustrate the extent of the problem caused by nodes that do not forward data, we looked into its effects when using a single dissemination tree with varying branching factors (number of children per internal node) and when using the multiple-tree SplitStream approach [17]. SplitStream is a robust and fair P2P system in which data is split into several slices and each slice is propagated through a different dissemination tree.

In the simulated experiments, a fixed percentage of the nodes do not upload any data to the system, while the rest of the nodes behave correctly, uploading



(a) Single Tree - Average Download Factor

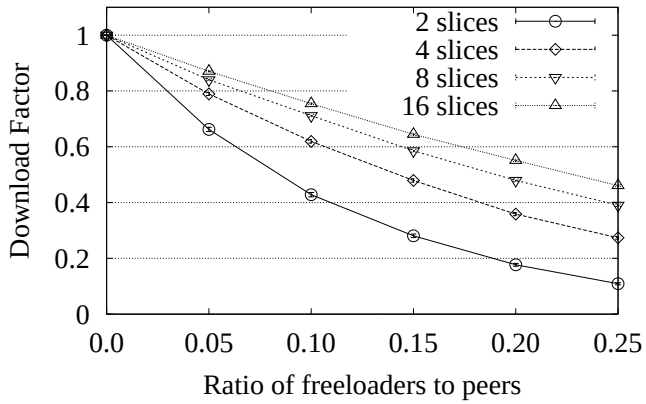


(b) Single Tree - Minimum Download Factor

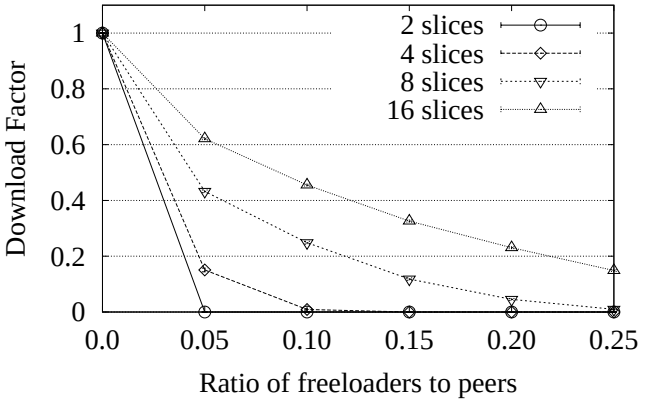
Figure 3.1: Effect of omission attacks on single-tree dissemination

as much as required by the protocol in use. As a measure of resilience, we compute the *download factor* of a streaming session, which is the ratio of packets received by a peer within the desired time to ensure proper playback. Using a round-based simulator, we computed the download factor of all nodes for sessions with a thousand homogeneous nodes and varying ratios of freeloading peers.

In Figures 3.1 and 3.2, we present the average and minimum download factors across correct nodes. We evaluated singles trees with varying branching



(a) SplitStream - Average Download Factor



(b) SplitStream - Minimum Download Factor

Figure 3.2: Effect of omission attacks on SplitStream

factors (BF), and SplitStream with varying number of slices. In the case of single trees, as observed in Figure 3.1(b), individual nodes are prevented from receiving any packet even with as little as 5% malicious members.

By splitting the dissemination of data into multiple trees, the SplitStream protocol presents better resilience to freeloading, since every node receives substreams of packets from a diverse set of peers. We varied the number of slices from 2 to 16. As expected, the larger the number of distinct dissemination trees used to propagate data, the less disruption observed by users. However, the

degradation observed by individual nodes is significant even with 16 slices (Figure 3.2(b)), when some nodes receive less than 80% of the data if 10 or more percent of nodes do not forward data.

In this chapter we present the design and evaluation of SecureStream, a system that combines techniques for improving the resilience of streaming against malicious behavior [44]. Our model of the system assumes the existence of one source, assumed non-compromised, disseminating data at a fixed rate to a set of receivers with limited buffering capacity. All nodes have similar download and upload capacities, slightly larger than the download rate. The desired behavior is that the streamed data be received within a fixed latency relative to the source's original transmission.

SecureStream combines a set of techniques to improve the resilience of streaming, which are described in detail in the next Section. SecureStream employs a pull-based streaming protocol, given its superior resilience to churn and simple failures when compared to tree-based streaming protocols (as described in Chapter 2). Furthermore, SecureStream employs an intrusion-tolerant membership protocol (*Fireflies*) to tolerate attacks to the membership layer. This protocol organizes nodes into a connected graph structure that defines each node's set of neighbors with which it may exchange packets. Such an imposed structure limits the extent of damage inflicted by high-level denial of service attacks and omission attacks, since attackers may not arbitrarily interact with any random node. Finally, SecureStream employs a simple yet efficient technique to avoid forgery of packets. We next describe these approaches in more detail and evaluate SecureStream both through simulation and execution of our SecureStream implementation on an emulation environment.

3.2 System Design

3.2.1 Pull-Based Streaming

SecureStream employs a pull-based approach to disseminate packets, following ideas used in the Chainsaw protocol [76], which is described in this Section.

Nodes are organized into a graph structure, which is used to define which nodes are allowed to exchange packets with one another. In the graph, each node has an approximately equal number of neighbors. We will describe how the graph is constructed in Subsection 3.2.2. The source of the stream is also assigned a random set of neighbors in the graph. Figure 3.3 shows an example of the connections between nodes in a small streaming system.

Initially, the source sends notifications to its neighbors as soon as it has available packets to disseminate. These notifications are small messages used to inform neighbors of availability of packets. Each neighbor requests packets from the source in a random fashion (based on the received notifications), to avoid overloading the source. As nodes receive packets, they propagate notifications to their neighbors, and so packets get disseminated along the system. This pull-based approach to acquisition of packets leads to increased resilience, since failure or misbehavior of one neighbor does not impede a peer from fetching packets from other neighbors. The predetermined set of neighbors for each node makes it hard for attackers to target individual nodes, since attackers lack deterministic means of acquiring control of all of a node's neighbors.

Pull-based streaming is simple and yet highly resilient to failures and attacks. The overhead incurred by notifications is not significant if large packets

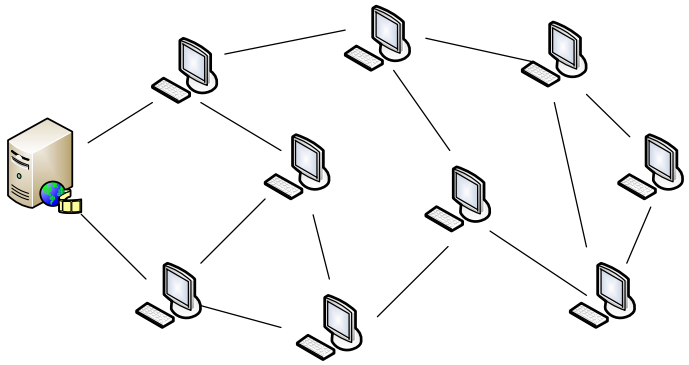


Figure 3.3: Connectivity between source and nodes in SecureStream

are used, and the protocol avoids receipt of duplicate packets. Since it is completely decentralized, the protocol does not present any single points of failure, another important consideration when building an intrusion-tolerant streaming protocol.

Timing Aspects

Packets are ordered by the source and each packet is assigned a sequence number. Each member stores packets and forwards them to other peers while the packet sequence number is within its *availability window*, which is a sliding window of sequence numbers that is updated as time changes. It also maintains an *interest window*, smaller than the availability window, which represents the set of packets in which the peer is currently interested. A node only requests packets which are within its *interest window*. Different policies can be employed by peers about what packets to pick from each of its neighbors, and the choice of the appropriate policy is crucial to achieving best overall performance. Random selection of neighbors is usually a good candidate, leading to fair load balancing, and is the one we use in our approach.

There is a predefined limit l on the number of outstanding requests to any neighbor. This policy is essential to maintain a good flow of packets in the system, avoiding nodes from overloading particular neighbors with requests, and ensuring that data doesn't follow a trivial tree-like path through all nodes. Furthermore, it also makes it harder for malicious peers to over-request packets from their neighbors. Peers maintain a queue of non-satisfied requests for packets, and if more than l requests by the same neighbor are present in the queue at any time, only the l most recent ones are maintained and potentially satisfied. This also reduces opportunities for malicious nodes to flood neighbors with packets: a node will never have over l requests for packets with any given neighbor, and therefore any burst of packets will be an indication of malicious behavior. The use of the limit l on the number of outstanding requests also provides a solution to network congestion, since nodes will only request further packets from neighbors whose connections can satisfy their requests in a timely manner.

3.2.2 Intrusion-Tolerant Membership Protocol

SecureStream relies on the *Fireflies* protocol [55] to define the communication graph while avoid membership attacks, through which attackers may attempt to control the set of neighbors with which peers interact. *Fireflies* is a scalable protocol for supporting intrusion-tolerant P2P systems, which provides correct nodes with a reasonably current view of all nodes which are live.

Fireflies is a one-hop membership protocol (every node knows about every other node) and is composed of three sub-protocols: a pinging protocol is used

to detect failures of nodes with an accuracy independent of message loss; an intrusion-tolerant gossip protocol is used for dissemination of information between correct members with probabilistic time bound Δ; and nodes use accusations and rebuttals to implement the membership information that *Fireflies* provides. These components are briefly described below. For a more comprehensive description, please refer to [55].

Each member monitors a specific set of peers for failures using an adaptive pinging protocol (we later describe the protocol used to define the set of monitored nodes). Members do not use a static global timeout when waiting for the replies of ping messages, but rather estimate the probability of message loss and try to adapt to the message loss characteristics between the monitor and monitored node.

Members are organized into multiple rings, and their position on each ring depends on their identifier. A hash function is associated with each ring, and this hash function is applied to each node's identifier to define the node's position on the ring. These rings determine which nodes monitor, and are allowed to accuse, which other nodes. On each ring, each member m_i monitors the lowest ranked successor m_j that it believes to be live, and if it detects a failed node, it issues an *accusation* for that node. In Figure 3.4, for example, node A monitors nodes B, D and F, while it is monitored by nodes E, F and G. The same idea is used to define neighborhood for the purposes of the pull-based streaming protocol described in the previous Subsection: each node exchanges packets with its immediate neighbors in each of these rings; nodes are therefore organized into a connected graph.

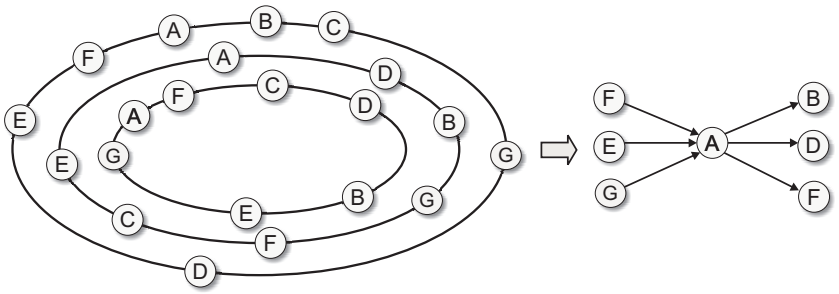


Figure 3.4: Logical rings in *Fireflies*

Members exchange two types of data structures in the *Fireflies* protocol: notes and accusations. All notes and accusations are signed, and a certification authority is responsible for issuing private/public key pairs and public key certificates. Notes are issued by a member to inform and convince other members that it is live. Accusations, as previously noted, are used by members to accuse other nodes of having failed. Notice that accusations are used only for indicating that a node has failed. Since each node has a determined set of neighbors, other problems such as flooding attacks and other similar nuisance attacks are avoided by nodes by ignoring the peers inflicting such attacks.

When an accusation for a member m_i is received by a member m_j, m_j waits a time period of length 2Δ, and then removes m_i from its view if the accusation is valid. This time period is established so that an accused member may issue a new *note* (a *rebuttal*) to an *accusation* against itself. In order to avoid malicious nodes from abusively accusing its correct neighbors in the rings, nodes may invalidate up to t rings, implying that accusations issued by its neighbors on those rings will not be accepted as valid by any correct member.

The dissemination of information such as accusations and rebuttals is performed using a robust gossip protocol. Each member periodically picks a random member from its view to exchange state information. The multiple ring

structure induces a gossip mesh resilient to malicious attacks. Gossip is highly robust and efficient, with probabilistic bounds on the latency of delivery to all members [60].

3.2.3 Ensuring Integrity of Data

One second important aspect which needs to be satisfied in a streaming session is that the data being distributed should not be tampered. Several authentication protocols have been employed for the general multicast paradigm, originally intended for IP Multicast [107, 39, 70, 94, 79]. The standard point-to-point mechanism of appending a message authentication code (MAC) computed using a shared key does not meet the security requirements of a multicast session. If receivers and sender share the same key, any receiver would be able to forge messages. On the other hand, signing every packet using a traditional asymmetric cryptographic protocol induces high overhead, and is therefore not desirable.

Live streaming systems present the following characteristics: packets are not all available to the sender prior to the transmission; receivers may use the data as soon as it is received, and receivers might not receive all packets. In this section, an overview of protocols that can potentially be used with these systems is presented. The first trivial approach, consisting of signing and verifying every packet does not require further description.

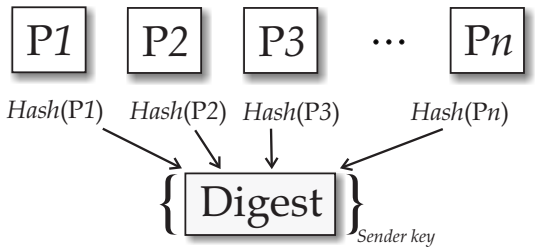


Figure 3.5: Linear digests authentication protocol

Linear Digests

To avoid the costs of signing and verifying every packet, it is possible to group the hashes of n packets into a special message, and have it signed by the sender (Figure 3.5). This signed message would need to be sent to the receivers prior to the dissemination of data that it corresponds to. This implies a buffering of content at the sender prior to the dissemination of data. The advantage of this approach is that it incurs a minimum network overhead of one hash per packet, while it amortizes the cost of a single signature/verification operation over n packets.

A special mechanism should be employed to guarantee that the signed message is received by all receivers. In order to minimize the buffering delay at the sender, small groups should be favored. Grouping packets every second or every few seconds should be satisfactory, since it is affordable to perform one signature/verification operation per second. This approach may lead to buffering at the receivers, if they do not receive the signature packet prior to the other packets in the group.

Merkle Tree Approach

Wong and Lam [107] amortize the cost of signature/verification operations over many packets in a stream. The sender computes the hashes of a limited number n of consecutive packets in the stream, and uses them as leaves in a *Merkle Tree*, which is a tree where each internal node consists of the hash of its children. Each leaf node contains the hash of a packet.

Each packet is appended with the root of the tree signed by the sender, the position of the packet's node in the tree, and all the siblings of the nodes on the path from the root to the node that represents the packet. This allows each packet to be verified without further delay. In Figure 3.6, for example, the shaded nodes indicate digests that should be appended to packet 5 for its transmission.

When a receiver first receives a packet from the group it uses the hashes of the nodes contained in the packet to compute the root of the Merkle Tree. The receiver can then verify the authenticity of the packet by using the signed hash of the root sent with the packet. Once the packet has been verified, the root does not need to be further verified for any other packets in the group. Therefore, for each group of packets, the computational cost consists of only one signature for the sender and one verification for each of the receivers.

The bandwidth overhead of this approach is 1 signature + $O(\log n)$ hashes per packet, since hashes of the node's siblings on the path up until the root need to be included with each packet. Finding the best tradeoff between computation cost and network overhead is important when deciding the number of packets to group in a Merkle Tree.

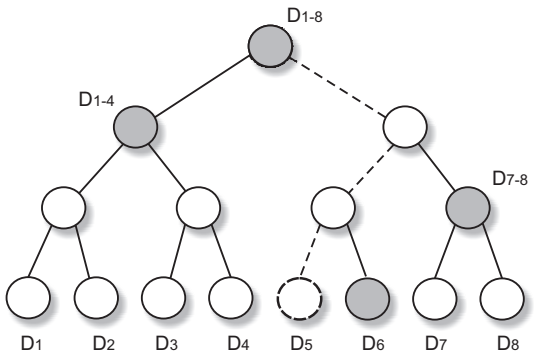


Figure 3.6: Merkle Tree chaining authentication technique

TESLA

In [79] Perrig et al. employ a delayed disclosure of keys approach, in which the sender attaches to each packet a MAC computed with a key originally known only to itself. Receivers cannot immediately authenticate packets upon receipt, and may accept or not packets based on the time of receipt. Some limited time d later, the sender attaches the key to a posterior packet, allowing the receiver to authenticate the buffered packet. The protocol assumes that all nodes are loosely synchronized with respect to the sender.

One important step of the approach is that receivers, upon receipt of a packet, should check that the key used to compute the MAC is still secret by determining that the sender could not yet have reached the time interval for disclosing it. Only if the MAC key is still secret should the receivers accept and buffer the packet. The key for a packet P_i is sent in a later packet P_{i+d}.

To avoid unverifiability of packets due to packet loss, a one-way chain is used in the reverse order of generation. One-way chains have the nice property that if packets are lost, they can be recomputed using later values in the chain. Given an initial key K_0, the sender uses a one-way pseudo random function

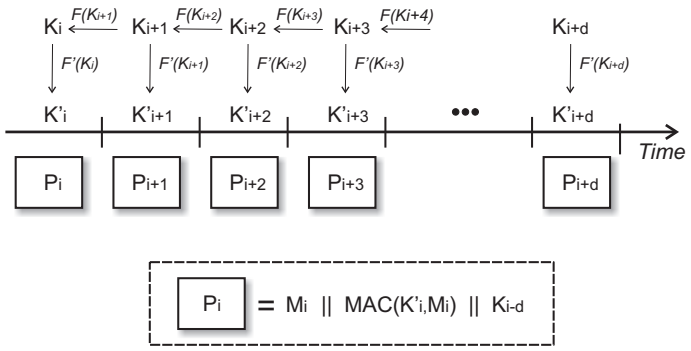


Figure 3.7: TESLA authentication

F to compute the next key in the chain. To avoid using the same key both as a generator to the next key in the chain and as a key for computing MACs, a second function F' is used. Given any key K_i, F' is used to derive the key to compute the MAC of packet P_i (Figure 3.7). Sender uses one-way function F to compute the set of keys K, and one-way function F' to compute the set of keys K'. A key K'_i is used to compute the MAC of packet i. Packet P_{i+d} contains key K'_i which allows authentication of packet P_i.

Prior to the transmission of a group of packets, the sender should transmit the time interval schedule, the key disclosure delay d and a key commitment to the key chain K_i over an authenticated channel (for example, using a digitally signed message).

The network overhead of this approach is one MAC per packet and there is no expensive signature/verification mechanisms involved, except in the beginning of the transmission. It is important to notice also that TESLA does not provide non-repudiation. Any node can forge authentic packets after the key is disclosed by the original sender.

Graph-based Authentication

Graph-based authentication was first employed in [70]. The idea consists of requiring that the sender sign only one packet. Subsequent packets in the stream are linked to that packet in a way that allows them to be verifiable. In order to tolerate packet loss, a graph is used instead of a single chain. Packets are represented by vertices in the graph, and a directed edge between nodes that represent packets P_i and P_j indicates that packet P_j contains the hash of packet P_i. A packet corresponding to a node can be authenticated if there is a path of already verified packets between the node and the signature node.

The main challenge in this approach is to build a graph that maximizes the probability that given the receipt of any packet, there will be a path from the node that represents it and the signature node so that the packet is verifiable. In [70], the authors employ a p-random graph, in which a directed edge between a node P_i and a node P_j is added with probability p.

In [94], Expander Graphs are used for the construction of graphs with a low constant degree independent of graph size. Their main goal is to minimize the bandwidth overhead and provide a lower bound on the probability that a packet can be authenticated upon arrival. Many random graphs are expander graphs and therefore present the properties described in the paper. In [80] Perrig et al. explore the use of random graphs and present simulation results indicating they are in general good candidates for use. However, they also show that not all random graphs yield good properties and some care needs to be taken to ensure proper graph construction.

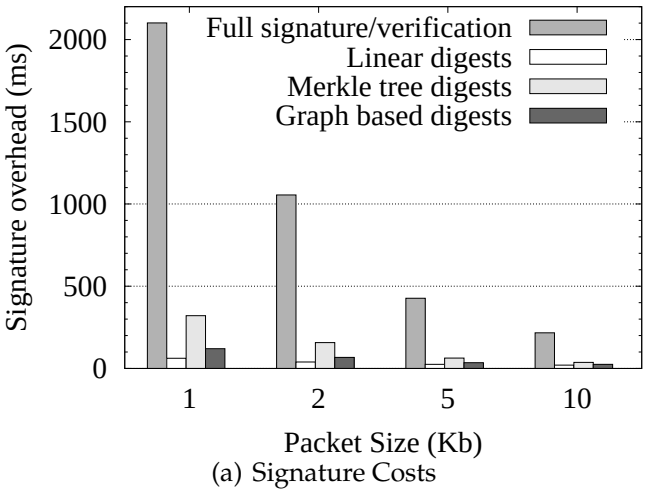
In a streaming protocol where packets are received out of order, graph-based authentication might require buffering of packets at the receiver.

Comparison of Protocols

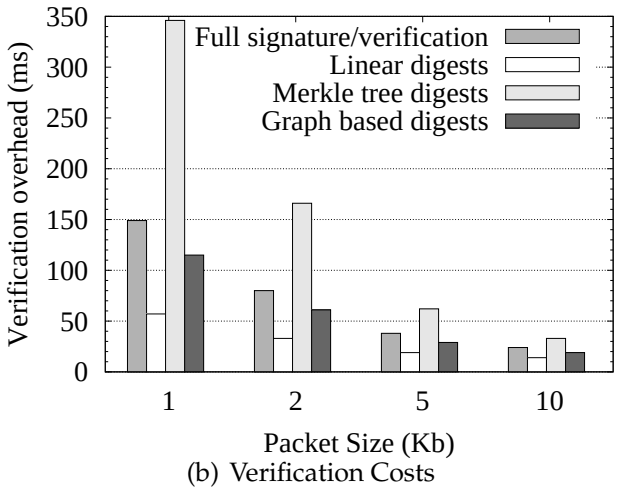
We aim to compare all the presented protocols in the context of peer to peer streaming, and use the Chainsaw style of dissemination as a case study for our comparison. We will consider the tradeoff between computational cost at the sender and receivers, network overhead, latency incurred in the system and guarantee of verifiability of packets. A fixed transmission rate of 300 KB/s is used and we consider the use of varying packet sizes, as well as different sizes of windows (buffering at the sender) for computing digests. The size of packets cannot be increased without limits since it influences the throughput of packets in a system like Chainsaw, where nodes fetch packets from multiple neighbors.

The computational costs per second at the sender and at the receiver for different authentication approaches are presented in Figure 3.8. Costs when using different packet sizes are presented, with larger packets presenting lower overheads, since fewer packets (and consequently fewer signatures) are required. The computational cost at the sender can be decomposed into a few components depending on the approach, including time to compute hashes, time to sign data and time to create digests in a format that can be transmitted. The cost at the receiver can be decomposed into time to compute hashes, time to verify signed data and time to parse the digests.

Figure 3.9 presents the network overhead incurred by the different approaches. As expected, the network overhead of the Merkle Tree approach is



(a) Signature Costs



(b) Verification Costs

Figure 3.8: Effect of packet size on signature and verification overheads

the highest, since one signature and a few hashes need to be appended to each packet in the flow (the size of one signature corresponds to approximately 6 hashes when SHA is used for hashing and RSA with 1024 bit key is used for authentication).

In a live streaming scenario, not all approaches yield full verifiability of received packets since not all packets are guaranteed to be received by every peer. Signing and verifying every packet has the advantage of guaranteeing full verifiability of received packets. Despite its higher costs, the Merkle Tree digest

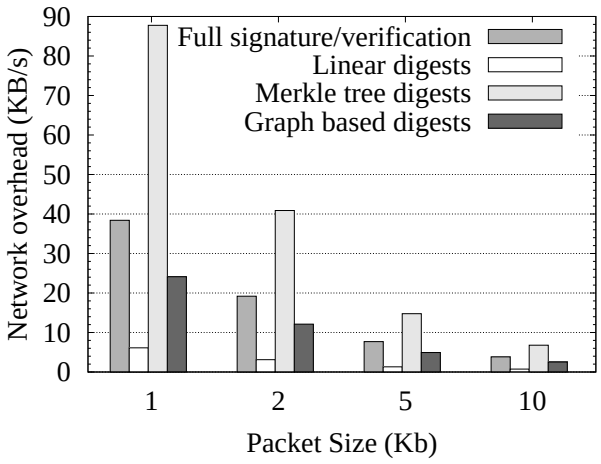


Figure 3.9: Effect of packet size on network overhead

approach is the only other approach that allows packets to be immediately verified upon receipt, even in the presence of packet loss. With the linear digest approach and the graph based authentication approaches it is possible that the necessary information is not available to the peers at the time of receipt of a packet.

The pull style of dissemination brings out certain problems in some of the presented techniques. Approaches which require buffering at the receiver, for example, are subject to denial of service attacks, since the receiver cannot filter the received packets before receiving authentication information. Another problem is that techniques which do not allow for immediate verification of packets at the receiver compromise the quality of the transmission, both in latency and in throughput.

Given its low costs, SecureStream employs the linear digests approach and ensures that verification is possible at the time of receipt of packets. We employed the following adaptation to the pull-based streaming protocol described in Subsection 3.2.1 to ensure immediate verifiability: each peer, when request-

ing a packet from a neighbor, uses a special bit in the request message that indicates whether the digest packet containing the hash for that packet has already been received or not. Since digest packets are small, they can be appended to the packet sent in reply to the request, ensuring that packets are immediately verifiable, only incurring a small overhead due to duplicate digests.

3.3 Evaluation

We originally evaluated the resilience of pull-based streaming in the presence of attacks through simulation. We also implemented SecureStream using Python, and we validated the simulation results by running experiments with our implementation on the Emulab testbed [106]. Emulab is a network testbed containing hundreds of nodes, in which real applications may be executed and evaluated. It allows arbitrary network topologies to be specified, leading to a controllable and repeatable environment.

3.3.1 Simulation

We built an event-driven simulator and simulated 200 node networks with 50ms inter-node latency. It would be possible to simulate and present results for networks with larger numbers of nodes, but a set of experiments on increasing numbers of nodes revealed that the behavior remains the same for networks as large as 5000 nodes. We opted for a smaller size but repeated each experiment 100 times to obtain better confidence in our results.

The target streaming rate in the experiments was fixed to 300 Kb/s, and packets of 10 Kb were used. Higher streaming rates yielded similar results as long as the packet size is accordingly increased to maintain a rate of 30 packets/s. Each streaming session lasted for 200 seconds. In the basic setting, the source's upload capacity was fixed to twice the streaming rate while other peers had a fixed maximum upload capacity of 1.2 times the streaming rate. These values are used as our baseline since they are the lowest upload rates at the source and non-source nodes respectively that lead to good throughput when the system is not under attack.

For each streaming session we computed the average and minimum download and upload rates across all correct members. We repeated each experiment 100 times, and we present the median and 95 percentile intervals across these repetitions.

We considered four types of malicious behavior. In the first type of attack malicious peers act faulty, neither requesting nor satisfying requests. In attack 2 they request packets but do not forward any packets. In attack 3 they over-request packets from their neighbors, requesting as many distinct packets as possible from every neighbor. Finally, in attack 4 they over-request packets and do not forward packets. The fourth type of attack is the most disruptive type and therefore the most likely, while the other three are considered mainly for comparison purposes.

Figures 3.10, 3.11, 3.12 and 3.13 present results for the basic setting under each of the attack types. We are interested in minimizing the overall damage to the streaming session. Damage is quantified by the impact on average down-

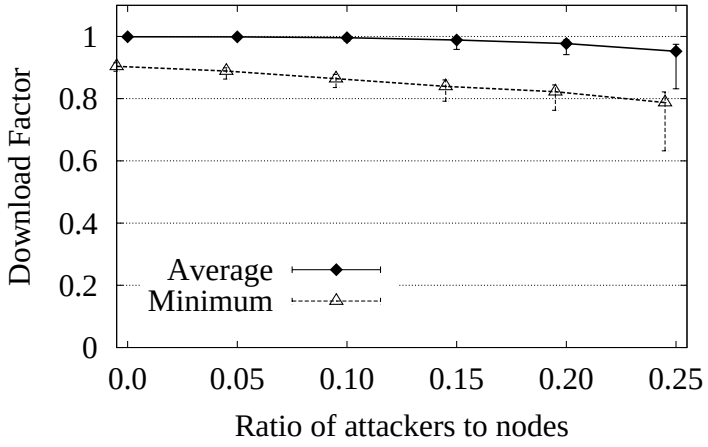


Figure 3.10: Effect of peers that completely fail

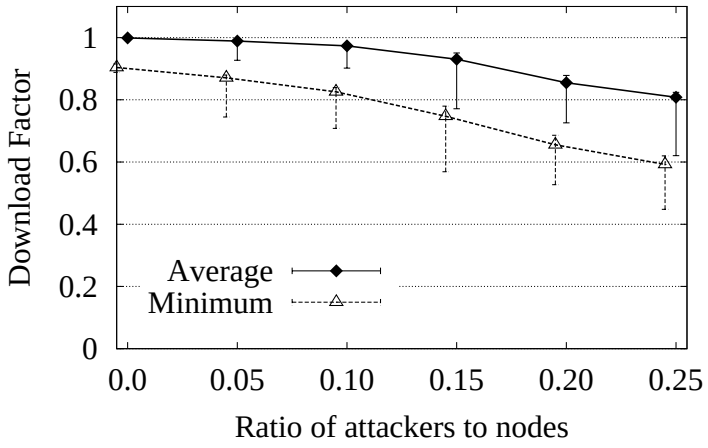


Figure 3.11: Effect of peers that do not forward packets

load rates to healthy nodes, and the minimum download rate for any single healthy node.

As would be expected, the results show that peer failure does not significantly affect the download rates since peers can still request packets from other correct neighbors (Figure 3.10). Since malicious peers do not request packets in this mode, they do not disrupt the total overall upload capacity. Even though upload rates are limited, over-requesting attacks are also not significantly disruptive, due to the random policy used by peers when satisfying neighbors'

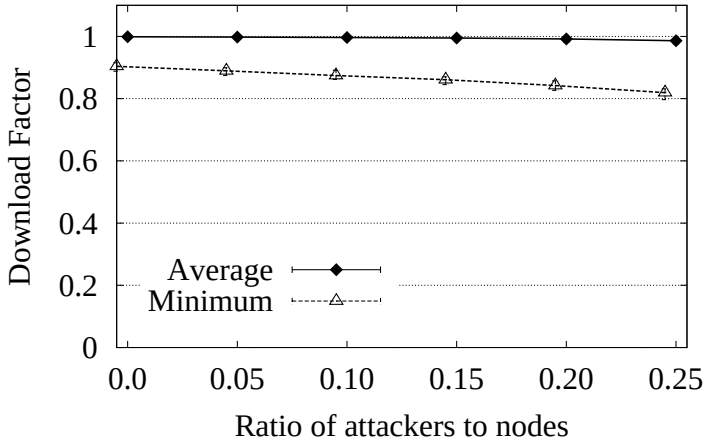


Figure 3.12: Effect of peers that over-request packets

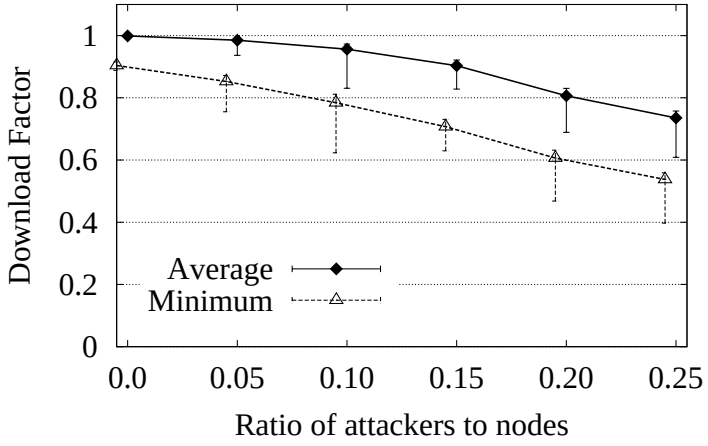


Figure 3.13: Effect of peers that over-request and do not forward packets

requests for packets and the upper limit on the number of outstanding requests by any neighbor (Figure 3.12).

Figures 3.11 and 3.13 show that attacks in which peers consume packets from their neighbors, but do not forward packets, inflict the most harm. There are two main reasons for this vulnerability. First, since peers upload at a maximum rate of 1.2 times the streaming rate, the overall upload capacity of the system gets compromised from peers consuming and not contributing to the system.

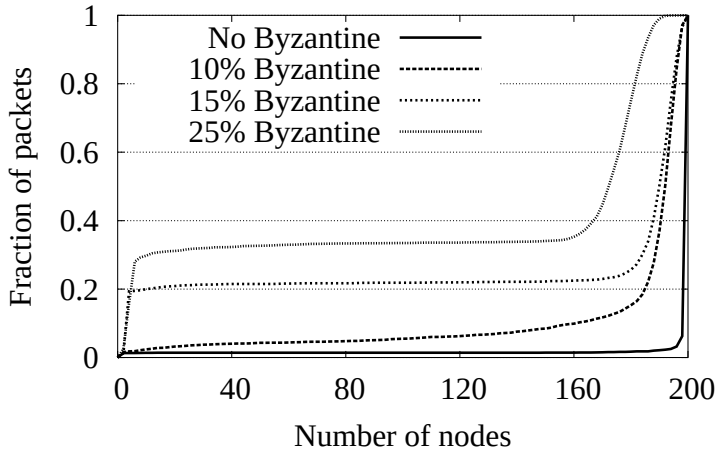


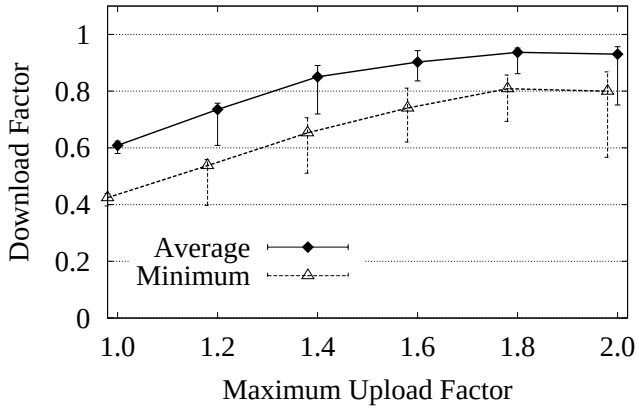
Figure 3.14: Cumulative distribution of number of packets received by nodes

Second, malicious nodes neighboring the source might impede some packets from ever being received by any other peer other than it.

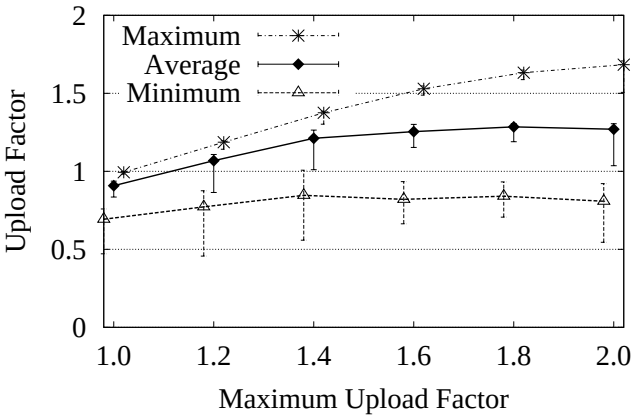
The latter effect causes the 95 percentile interval bars to be wide: there is a lot of variation depending on the number of compromised peers near the source. To make this point clear, in Figure 3.14 we show the percentage of packets received by increasing numbers of peers during sample streaming sessions with varying percentages of Byzantine peers. The metric to focus on here is the fraction of packets only received by one peer, which is an indicator of malicious nodes neighboring the source. Packets received only by malicious peers at the first hop will never be disseminated in the system. To confirm this hypothesis, we executed the same set of experiments and restricted the malicious attackers to being located at least 2 hops away from the source. The obtained medians were very close to the medians obtained in the previous experiments. The main difference was that the percentile intervals were significantly reduced when the source had no immediate malicious neighbor, which is an important result since the intervals are significant in the original experiments with attacks 3 and 4.

To improve the resilience, we can vary parameters to improve the overall upload capacity of the system, or to avoid situations in which malicious peers can isolate certain packets. First, we considered the upload capacity of the members. In Figure 3.15(a) we varied the value from 1.0 to 2.0 times the streaming rate and verified the improvements to resilience under attack 4. This graph presents the average and minimum download rates when the system has 25% of Byzantine members. The results show that the higher the upload capacity at non-source peers the more resilient the system becomes. From Figure 3.15(b), which presents the minimum, average and maximum upload rates of members, we can see that as a consequence of increasing the upload capacity of peers the system becomes more unfair, with an increased difference between the maximum upload rate and minimum upload rate across peers. For the next few experiments we fixed the upload capacity of non-source members to 1.4 times the streaming rate.

To improve the packet loss rate at the first hop from the source, we varied the upload capacity of the source from 1.0 all the way to 6.0 times the streaming rate. Our results indicated that this naïve approach to increasing the upload rate at the source does not significantly affect the resilience of the system. We also observed that the number of neighbors of the source is a more significant parameter than the upload capacity of the source. We fixed the percentage of malicious nodes at 25%, the upload rate at non-source nodes to 1.4 times the streaming rate and at the source to 4.0 times the streaming rate, and varied the source's number of neighbors from 4 to 20. The median slightly improves as the number of neighbors is increased, but more importantly, the percentile intervals are significantly reduced. In Figure 3.16(a) we present the absolute sizes of the 95 percentile intervals varying with the number of neighbors of the source. The



(a) Download Factor

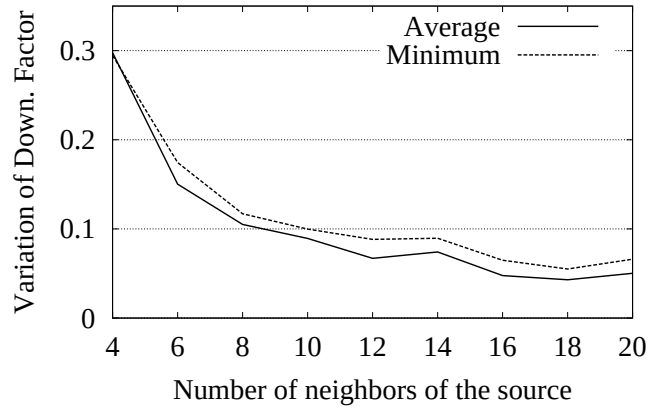


(b) Upload Factor

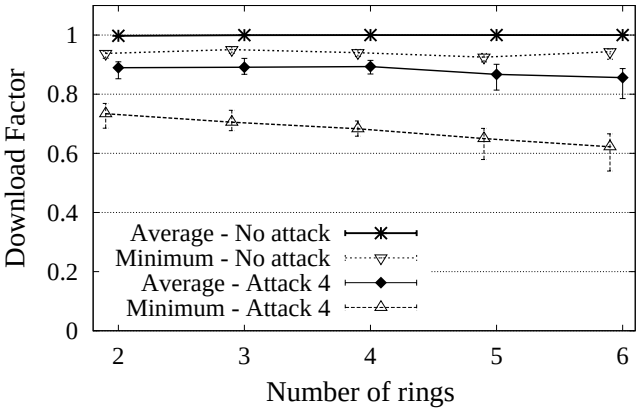
Figure 3.15: Sensitivity to peers' maximum upload capacity

results show that a larger number of neighbors at the source is desirable. This happens because with a higher number of neighbors the percentage of malicious neighbors of the source tends to be closer to 25% across runs, and therefore there is less variation in the percentage of packets that are contained at the first hop from the source.

Finally, to study the influence of the number of neighbors for each non-source peer in the system, we evaluated the resilience with a varying number of rings used to define neighbors. The upload capacities at the source and non-



(a) Source node



(b) All other nodes

Figure 3.16: Sensitivity to number of neighbors

source members were fixed to 4.0 and 1.4 times the streaming rate, respectively, and the source had 16 neighbors. In Figure 3.16(b) we present the performance of the system using between 4 and 12 neighbors per node, both under no attacks and under attacks of type 4. The results surprisingly show that the use of larger numbers of neighbors does not improve resilience of the system, and even reduces when the system is under attack. Even though larger numbers of neighbors would lead to better connectivity between correct members, it also presents malicious members with more potential to over-request packets and unbalance the system.

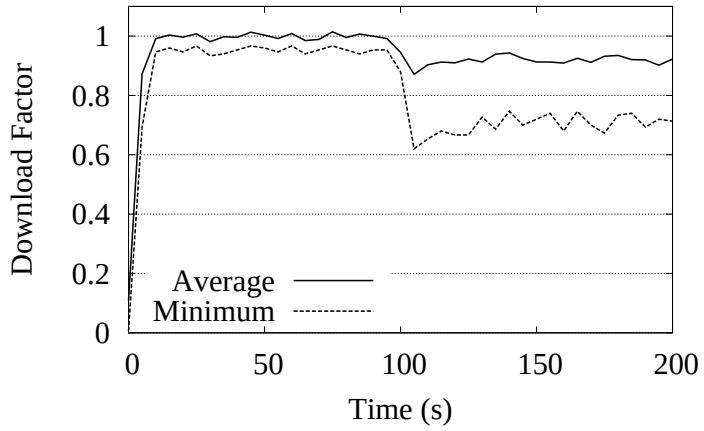
3.3.2 Emulation

In order to validate our simulation results, we ran experiments on a 200 node network on the Emulab testbed using our Python implementation of the SecureStream system. We performed extensive experiments under various parameter configurations, observing that the tendencies observed were similar to those verified through simulation.

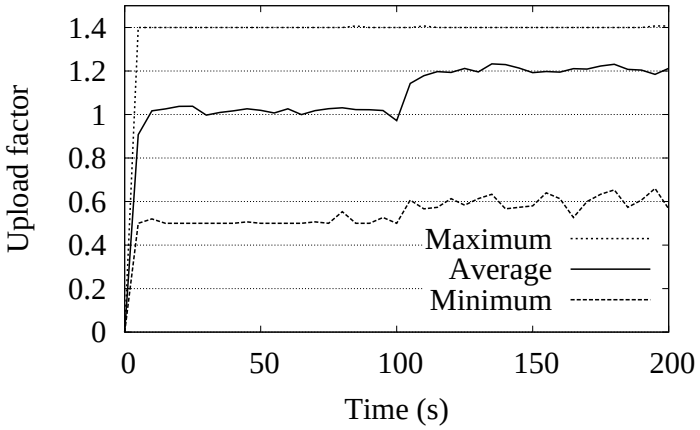
To illustrate the behavior of the real system in execution, we present results of a sample streaming session in which 25% of the nodes are malicious, over-requesting packets and not forwarding them to neighbors. During the first 100 seconds all nodes act correctly, after which the malicious nodes start over-requesting and not forwarding packets. We fixed the upload capacity of the source and non-source members to 4.0 and 1.4 times the streaming rate respectively, and the number of neighbors of the source and non-source members to 12 and 8 respectively.

Figure 3.17(a) presents the minimum and average continuity indices of correct peers throughout the sample session. Around the hundredth second, the average and minimum continuity indices decrease with the insertion of malicious peers. The minimum, average and maximum upload factors across all correct peers is presented in Figure 3.17(b). At the point when malicious nodes are inserted, the upload factors across correct peers increases to compensate for the malicious peers consuming the scarce resources from the system.

We also observed the effect of the system in the latency of packets. In Figure 3.18(a), a slight increase in the overall average and maximum delays per packet in the presence of attackers may be observed. Furthermore, an interest-



(a) Download Factor

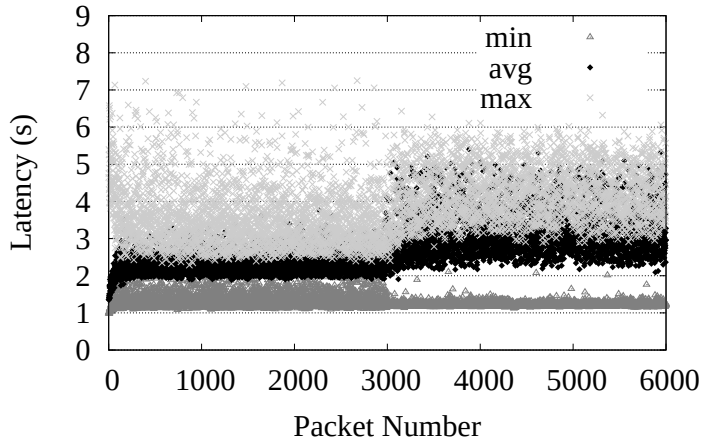


(b) Upload Factor

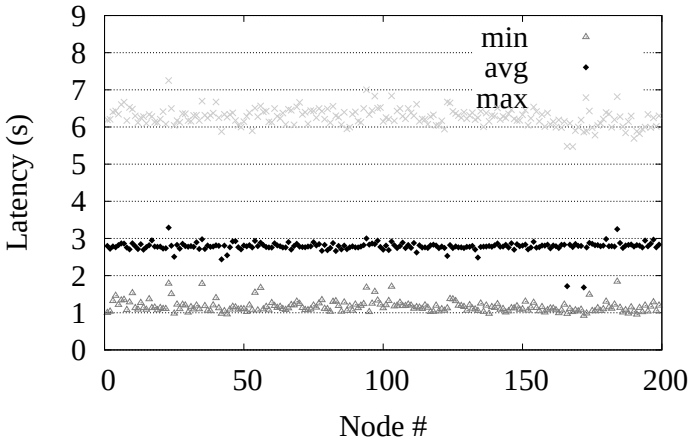
Figure 3.17: Sample streaming session on the Emulab testbed

ing behavior can be observed in Figure 3.18(b), which presents the minimum, average and maximum packet delays for each node in the system, relative to the time of origin of the packet at the source.

Unlike our initial suspicion, all nodes presented similar packet delays over the streaming session. This indicates that being close to the source does not imply in receiving packets faster than other nodes, since not all packets will be requested from the source, because of the limit in number of outstanding requests. To verify if this behavior might vary when the system scales to larger



(a) Latency Per Packet



(b) Latency Per Node

Figure 3.18: Latency of packets on the Emulab testbed

numbers of nodes, we simulated networks with up to five thousand nodes. The maximum latency used for bigger networks needs to be increased, but the average latency per node is still similar.

We also looked into the number of hops taken by packets before reaching all nodes. For the same streaming session, we registered the number of hops taken by each packet before reaching each node. In Figure 3.19 we present the CDF of the average and maximum number of hops taken by packets. This graph shows that for our sample session with 200 nodes, the average number of hops mostly

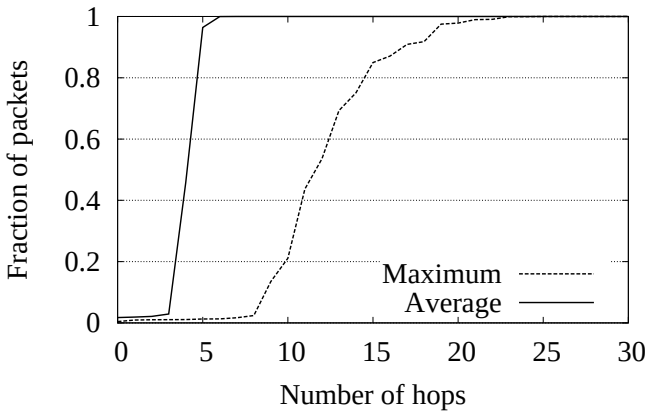


Figure 3.19: Cumulative distribution of avg and max packet hop count

varies between 3 and 5, while the maximum number of hops taken by each packet varies between 8 and 22. This large variation in the maximum number of hops is also verified by the large variation in maximum latency observed in Figure 3.18(a).

3.4 Summary

In this chapter we presented the design and evaluation of SecureStream, an intrusion-tolerant peer-to-peer live streaming system. We described and evaluated a set of techniques that allows SecureStream to resist against forgery, DoS, membership and omission attacks. We showed through simulation and emulation the effect of pull-based streaming on resilience to the class of attacks we investigated, and we explored how the variation of specific parameters affect the resilience of the system. Our results indicate that SecureStream tolerates a limited percentage of malicious nodes, gracefully degrading in the presence of increasing ratios of attackers to the total number of nodes.

CHAPTER 4

ENFORCING FAIRNESS THROUGH AUDITING

4.1 Introduction

The peer-to-per (P2P) paradigm allows systems to scale with the number of users, but also leaves them vulnerable to freeloading behavior. Freeloading nodes attempt to receive a stream without uploading their fair share of data, reducing the overall upload capacity of the system. Despite the damage they may cause, not much work has been done in studying mechanisms to avoid their presence in live streaming systems. In this chapter, a technique for defending against freeloading users without incurring high overheads is described.

The live streaming approach that most closely relates to our work and attempts to discourage freeloading is the BAR Gossip protocol [65], which employs a *tit-for-tat* approach for encouraging nodes to contribute: a node only sends as much data to another node as it receives back. BAR Gossip provides an elegant solution shown to tolerate both freeloading behavior and other malicious attacks. However, reliance on tit-for-tat does present a few undesirable requirements. To be efficient, the data source should ensure that packets are evenly spread across the system by sending data to a fixed proportion of nodes, and by sending different packets to different nodes. Furthermore, it requires the source and all nodes to have full membership knowledge. These restrictions affect scalability when the data source has bounded upload bandwidth.

To illustrate this problem, we fixed the upload capacity of a data source at 5 Mbps and simulated BAR Gossip when streaming 500 Kbps with increasing

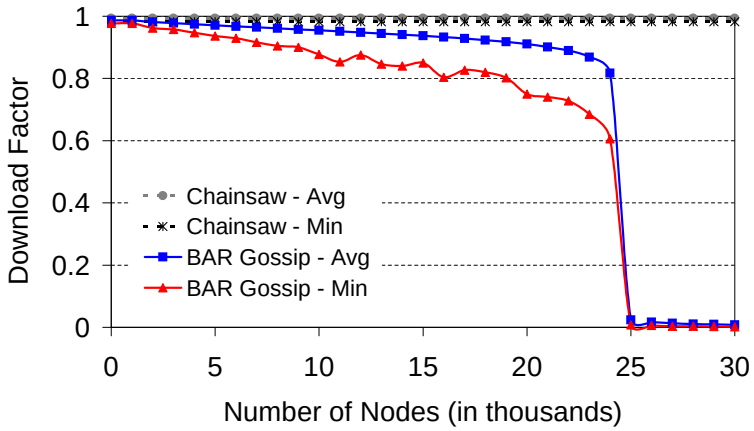


Figure 4.1: Scalability of BAR Gossip and Chainsaw protocols

numbers of receivers, varied between one and thirty thousand nodes. We compare its scalability against the Chainsaw protocol [76], for which we fixed the source's upload bandwidth to a lower value of 2 Mbps. In Figure 4.1, we present the average and minimum download factor (ratio of the download rate over the stream rate) of both protocols when the number of nodes is increased. As observed, BAR Gossip is not able to sustain its performance without scaling the upload capacity of the source proportionally with the size of the system. Meanwhile, Chainsaw is able to scale well even with a fixed lower upload bandwidth at the source, but cannot handle the presence of freeloading nodes.

We use auditing to encourage data-sharing in live streaming systems like Chainsaw. Our auditing approach establishes a minimum threshold for the amount of data sent by any node in the system, and removes nodes that upload less data than the threshold. Instead of relying on a tit-for-tat mechanism, we focus on encouraging nodes to respect the established protocol. Nodes are forced to provide verifyiable information regarding packets sent to and received from neighbors, and the auditing system is responsible for detecting and removing misbehaving nodes.

Notice that identifying the misbehaving nodes is not a trivial task, since there is no fixed minimum amount of data that nodes should contribute to the system. If we assume a model where misbehaving nodes simply did not upload any data, detecting them would be an easier task. However, once we assume that misbehaving nodes may adjust their contribution level based on the policy used by an auditing system, a more elaborate approach is required. This chapter presents and evaluates an auditing model based on sampling the system and using the sampled information to build a global view of how the system is currently behaving. Based on it, auditors employ strategies to identify the misbehaving nodes that should be punished.

4.2 System Model

Our approach focuses on a target streaming system consisting of one data *source* (assumed non-compromised), which disseminates data at a fixed rate to a dynamic set of receivers. The source has limited upload bandwidth, and hence can only send data directly to a small subset of interested receivers. Participating nodes are consequently required to forward packets to their neighbors, helping disseminate all packets across the system. The streamed data should be received by all nodes within a fixed latency from the source's original transmission, even in the presence of freeloading nodes.

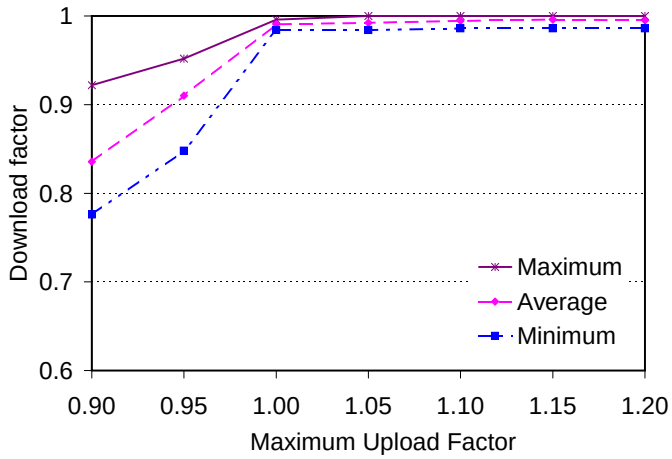
Our auditing approach is used over the Chainsaw protocol [76], previously described in Chapter 3 (Section 3.2). The streaming process starts at the source, which breaks the data stream into packets and sends notifications to its neighbors as soon as it has packets to disseminate. These notifications are small mes-

sages used only to inform neighbors of the availability of new packets. Based on the received notifications, each node requests missing packets, and the source satisfies as many requests as allowed by its upload capacity. Unlike BAR Gossip, with Chainsaw the upload capacity of the source does not need to increase with the size of the system; even an upload capacity of twice the stream rate is sufficient to ensure that the system performs and scales well. As nodes receive packets, they mimic the role of the source, sending notifications to their own neighbors in the mesh, allowing packets to be propagated through the system.

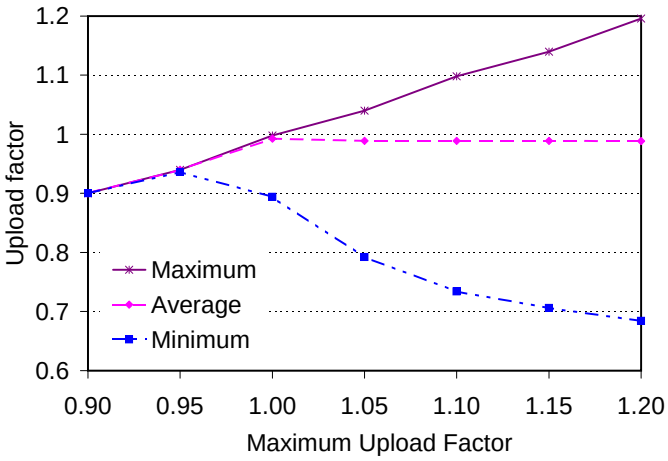
4.2.1 Expected Behavior

Our first goal is to explore the typical signature of the system, since an understanding of the behavior of pull-based dissemination in the absence of freeloading nodes will be important when we set out to introduce auditing. We conducted experiments using an event-based simulator, which is described in more detail in Section 4.4.

In Figure 4.2, we evaluate the performance of 1000 nodes during an ideal execution of Chainsaw, where all nodes behave correctly. We fixed the upload factor of the source at 4.0 (2 Mbps), and the stream rate to 500 Kbps. We varied the maximum upload factor of nodes to see how it affected both the download and upload factors of nodes across the system. The maximum upload factor is a fixed parameter which defines the maximum rate at which a node will upload data to all its neighbors. For fairness in nodes' bandwidth consumption, we would like all nodes to upload data at a factor as close as possible to 1.0. We varied the maximum upload factor of nodes from 0.9 to 1.2.



(a) Download Bandwidth



(b) Upload Bandwidth

Figure 4.2: Bandwidth usage when maximum contribution rate is varied

Figure 4.2(a) shows the minimum, average and maximum download factors across the nodes when the maximum upload factor of nodes is increased. As observed, by increasing the maximum upload contribution, we increase the global upload capacity of the system, leading to a better flow of packets. However, the discrepancy among the upload factors of individual nodes also increases, as seen in Figure 4.2(b). Some nodes participate more actively in dissemination while others end up contributing less, even though all of them behave correctly. This is an important consideration: when we introduce auditing, we hope not to

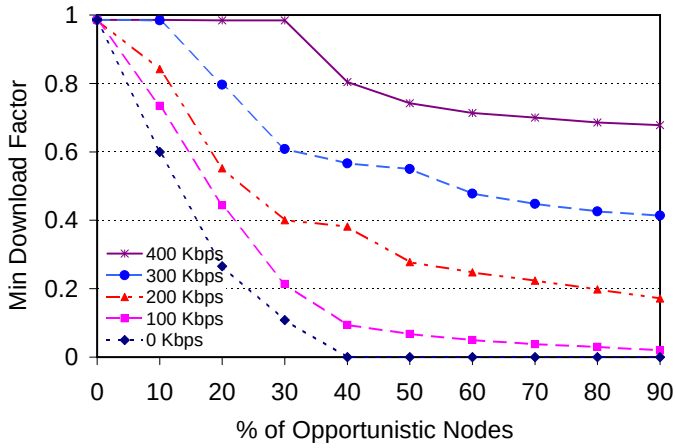
punish nodes that are willing to contribute but cannot do so because of factors such as their physical positioning in the system. In all our future experiments we set the maximum upload factor to 1.1.

4.2.2 Effect of Freeloading Behavior

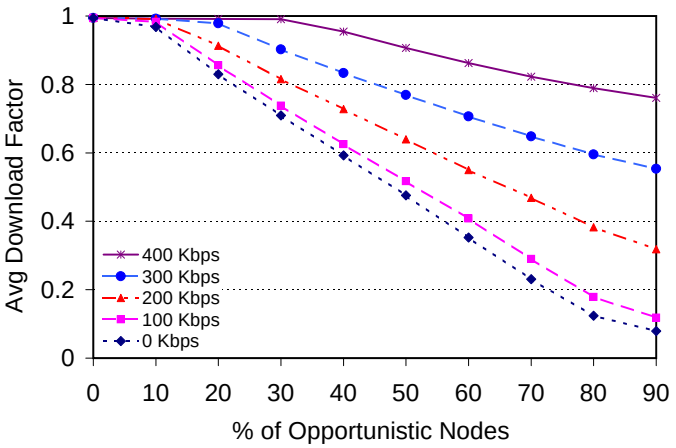
Our next goal was to understand the expected behavior of correct nodes under different scenarios where freeloading nodes compromise the system. We therefore studied how the download and contribution rates of correct nodes are affected under these conditions. Freeloading nodes may contribute with some data in an attempt to disguise their opportunistic behavior. Therefore, we considered different rates of contribution for freeloading nodes: 0 (pure freeloaders), 100, 200, 300 and 400 Kbps.

Figure 4.3 presents the average and minimum download factors among all correct nodes under different configurations. The stream rate was fixed at 500 Kbps, and all correct nodes had a maximum upload factor of 1.1 (550 Kbps). We ran experiments with 1000 nodes and increasing percentages of freeloading nodes in the system (from 0 to 90%). On the x-axis, we vary the percentage of freeloading nodes. As expected, we can observe that the download factors of correct nodes decrease since the aggregated upload capacity in the system becomes insufficient to provide all nodes with all data. Nonetheless, the extent of the impact may be surprising: with just 10% freeloading nodes, performance drops by as much as 40%.

Figure 4.4 presents the average and minimum upload factors among all correct nodes. Once again, on the x-axis we vary the percentage of freeloading



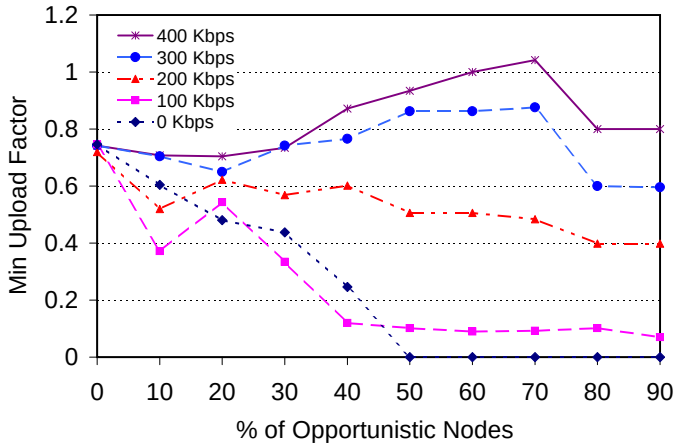
(a) Minimum Download



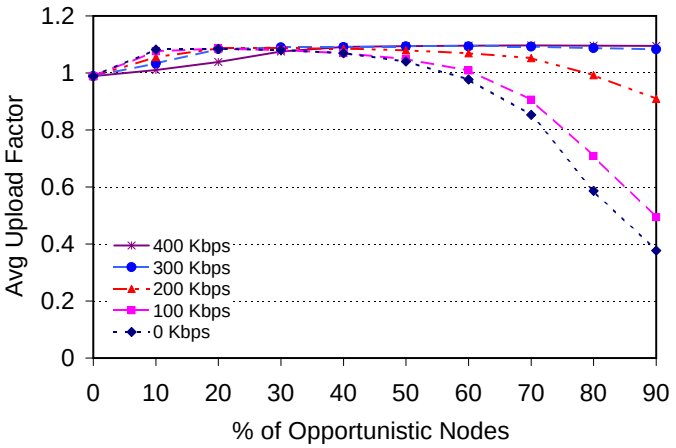
(b) Average Download

Figure 4.3: Effect of freeloading on peers' download rates

nodes, and on the y-axis we present the upload factors of nodes, which can vary up to 1.1. It is interesting to note that the average upload factor among correct nodes initially increases, and then starts falling when the percentage of freeloading nodes increases significantly. This behavior can be explained by the fact that, initially, correct nodes start contributing more to compensate for the lack of data provided by a small percentage of freeloading nodes; however, once the effect of freeloading nodes becomes significant, the system collapses and correct nodes are not able to keep contributing.



(a) Minimum Upload



(b) Average Upload

Figure 4.4: Effect of freeloading on peers' upload rates

Another important point to note is that the minimum upload factor does not follow a clearly defined pattern, making it hard to estimate the minimum contribution of correct nodes under compromised scenarios. Therefore, by applying thresholds to punish freeloading nodes, correct nodes may also be unfairly penalized.

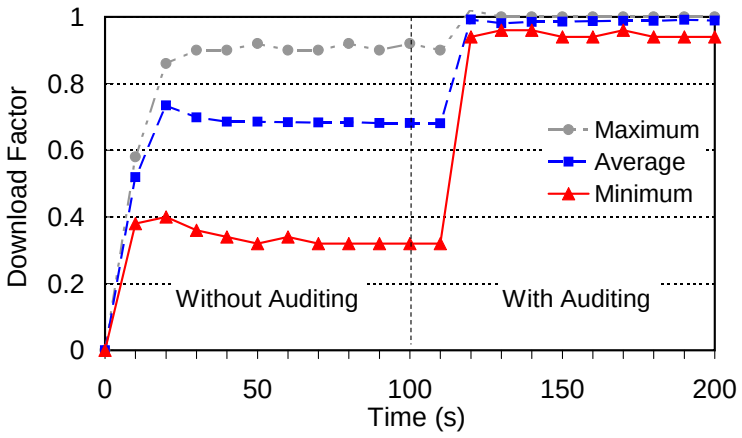


Figure 4.5: Use of auditing on a sample streaming session

4.3 System Design

Our idea for auditing the described live streaming system against freeloading behavior is motivated by the graphs presented in the previous section: we employ auditing to ensure that all nodes in the system contribute more than a particular specified threshold. In Figure 4.5, we illustrate the potential benefit from using auditing in a system where 70% of the nodes are correct and 30% are freeloaders. The latter do not upload any data. During the first 100 seconds, no punishment was applied in an attempt to simulate a system with no auditing. At time t = 100s, auditing is enabled and freeloading nodes start to be expelled from the system for low contribution. For this experiment, the minimum upload factor for nodes to stay in the system was set to 0.5.

We present the minimum, average and maximum download factors across correct nodes varying along 200 seconds. As observed in this particular example, auditing has the potential to improve the quality of streamed sessions significantly, and at low cost. One important concern is that if the specified threshold is too high, more freeloading nodes may be caught, but correct nodes

may also be unfairly punished. In this experiment, no correct nodes were mistakenly expelled from the system.

4.3.1 Auditing components

We now give some additional details of the auditing architecture, focusing upon two aspects: (1) collecting auditable information about the download and upload factors of individual nodes in the system; and (2) establishing the best threshold at any given time during execution and removing nodes that do not contribute at least as much data as specified by the threshold. We employ two types of components to perform these two roles: local and global auditors. Local auditors are executed on the nodes participating in the system, and therefore cannot be trusted; if a node is malicious, it might report false data. Global auditors are trusted components that run on dedicated external nodes. There can be just one or a few global auditors. We describe their roles and interactions in detail below.

Local Auditors

Each node n runs a local auditor, which interacts with other local auditors and has two main roles:

Publish n's data exchange history: n's local auditor periodically compiles and distributes the history of packets exchanged by n. To accomplish this, every δ seconds, it queries the local streaming application running on n for the set of packets it sent and received using the streaming protocol in the

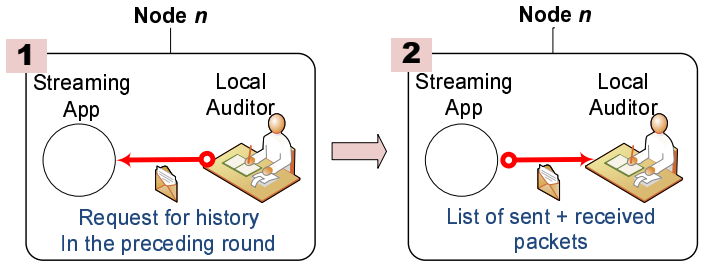


Figure 4.6: Local Auditing

most recent time interval (Figure 4.6). The local auditor signs and publishes the collected history to an assigned subset of its neighboring nodes, from which other auditors may obtain it. This level of indirection is used to prevent nodes from masking their real upload and download factors by presenting different information to different auditors.

Audit *n*'s neighbors' histories: *n*'s local auditor periodically audits the published histories of the nodes with which *n* exchanges packets. For instance, if node *n* exchanges packets with nodes *p*, *q* and *r* in the live streaming protocol, *n*'s local auditor compares these three nodes' histories with *n*'s own history. This involves ensuring that: (1) the amount of data sent by these nodes satisfies the defined minimum threshold for the system; and (2) the set of packets they claim to have sent to and received from node *n* corresponds to the set of packets *n* claims to have respectively received from and sent to them. If the first check comparison fails, the local auditor issues an *accusation* against the node to a global auditor. In the second case, the local auditor is not able to prove the neighbor's misbehavior; instead, it instructs its local streaming application to not further exchange packets with the misbehaving neighbor. More complex types of checks may also be performed to address other types of Byzantine behavior.

There are two ways in which a node could pretend to be sending more or receiving less data than it actually does. It could send different histories to each neighbor, always lying about its interactions with other neighbors. For example, n could send a history to p pretending to send more data to q than it actually did, while it sends a different history to q where it pretends to send more data to p than it actually did. n's goal would be to send less data while not being caught by any of its neighbors. The process of publishing a node's history to a predefined set of neighbors ensures that the node cannot send conflicting histories to different neighbors undetected, therefore avoiding this problem.

A node could also lie about the set of packets sent to or received from a particular neighbor p. In this case, p will be able to identify that the node has lied and will therefore stop exchanging packets with n. Given that a freeloading node's goal is to maximize its utility, it should have no interest in losing data exchange partners. Therefore, freeloaders have no incentive to publish incorrect histories.

Summary: Local auditing ensures that correct information is available regarding the set of data sent and received by any node, and allows nodes to monitor each other's contribution rates.

Global Auditors

Global auditors are trusted components with global membership knowledge, which interact with one another and with the local auditors. As shown in Figure 4.7, global auditors execute on nodes external to the system. Their main roles are:

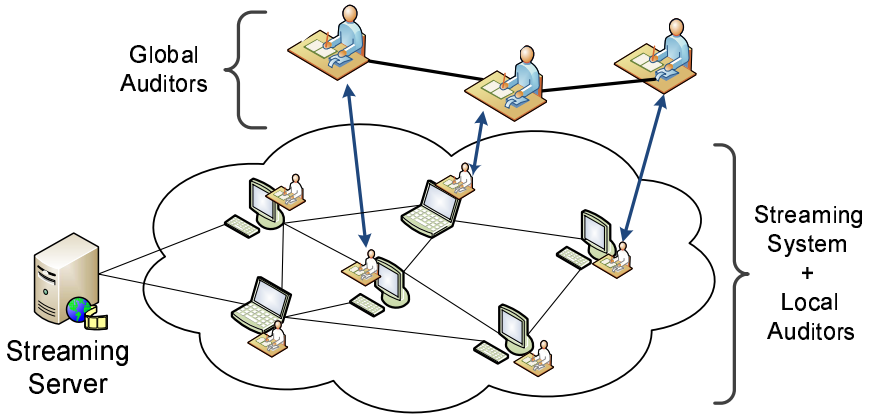


Figure 4.7: Global Auditing

Define the minimum upload threshold: Global auditors periodically sample the state of the system by querying local auditors. They then cooperate to analyze the collected samples, and on this basis compute the minimum upload contribution threshold. Different strategies may be employed for choosing the best possible threshold, given different scenarios. Global auditors may change the threshold based on the current conditions of the streaming session. If the threshold is modified, it is gossiped to all local auditors, which apply it when auditing their neighbors.

Remove nodes from the system: Global auditors are also responsible for verifying accusations issued by local auditors against particular nodes, and after validating the accusation, removing misbehaving nodes from the system. Validation involves verifying that the accused node's history indeed indicates that the node is sending less data than the current threshold. Removing a node involves informing the nodes' immediate neighbors of its status and forcing the removal of the node from the overlay mesh.

The number of global auditors may vary according to different parameters, such as the size of the system. The use of more global auditors distributes the load of sampling and improves efficiency in reacting to accusations against nodes. Global auditors are also perfect candidates to perform membership tasks such as acting as rendezvous points, since they are required to have full membership knowledge of the system for performing their auditing roles.

Summary: Global auditing monitors the global health of the system to identify the best value for the minimum upload threshold at any time during a streaming session, and makes final decisions regarding punishment of nodes.

4.3.2 Adaptive Threshold Strategies

Choosing an upload threshold requires care: a low threshold may not be sufficient to identify freeloading nodes, while high thresholds may incorrectly punish correct nodes. We considered three strategies for the choice of the minimum contribution threshold used for identifying misbehaving nodes.

The simplest strategy sets a fixed threshold (e.g., $t = 0.5$), independent of the current state of the system. In this case, any node contributing at a rate of less than 50% of the stream rate would be removed. One downside of using a fixed threshold is that freeloading nodes that learn the threshold can simply contribute at the lowest possible upload factor, thus avoiding detection. From the graphs in section 4.2, it is clear that such a strategy may disrupt the streaming session. Meanwhile, choosing a high threshold is not a practical option, since correct nodes would get unfairly punished.

To avoid this problem, we have explored adaptive strategies. One simple strategy starts with a minimum threshold (e.g., $t = 0.5$), increasing it only if the system is compromised. Global auditors sample the system to identify the average download factor, and if this factor is lower than 0.98, increase the threshold. Once the download factor reaches a satisfactory level again, the threshold may be reduced back to its initial value. This *stepwise* approach allows the system to catch freeloading nodes in case their presence starts affecting the performance of the system, while avoiding incorrect accusations of correct nodes.

We also considered a second adaptive strategy (*percentile-based*) for computing the threshold based on periodically sampled download and upload factors. The average download factors once again are used for detecting whether the threshold should be varied or not. In this strategy, our initial threshold is set to null. After each sampling, global auditors determine if the system is in a compromised state by verifying if the average sampled download factor is lower than 0.97 . If so, the threshold is set to be the 10th percentile of the sampled upload factors. This approach relies on efficiently sampling the system, and on the assumption that if the system's performance is not satisfactory, then at least 10 percent of the nodes are freeloaders.

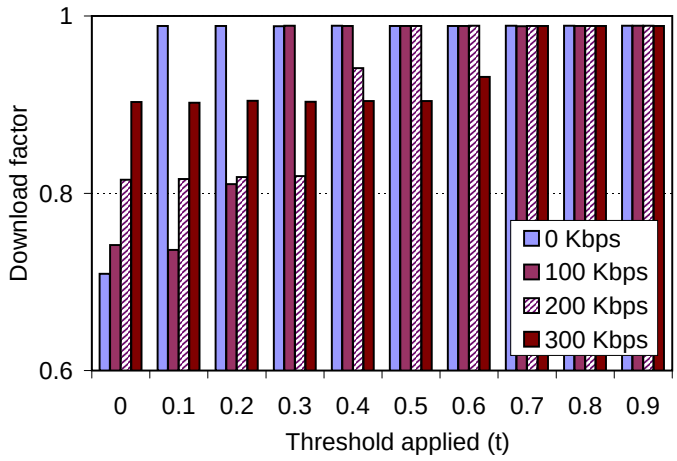
4.4 Evaluation

In this section, we evaluate the performance of our auditing strategy over the original streaming protocol. We built an event-driven simulator and used it to simulate streaming sessions on networks with 1000 nodes and an average of 50 ms inter-node latency. The target streaming rate in the experiments was fixed

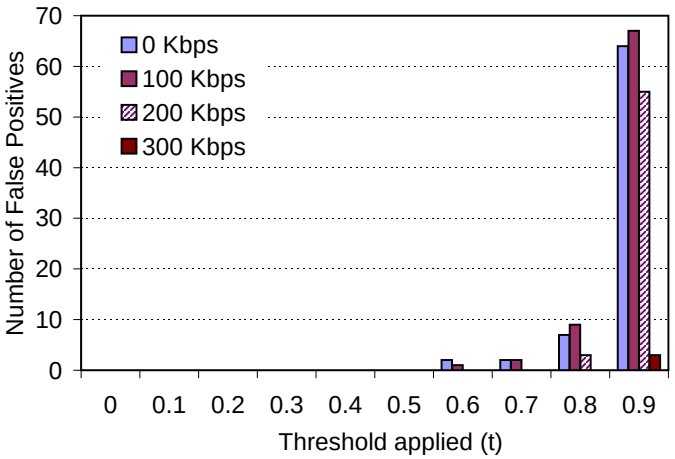
to 500 Kb/second, and all our experiments were repeated 10 times. Confidence intervals were small, and for simplicity are omitted from the graphs.

In all experiments, the source of the stream has an upload capacity of four times the stream rate (2 Mbps) and is connected to 20 arbitrarily selected nodes. Other nodes have enough download capacity to receive the stream, and upload factor of 1.1. We defined an availability window of 10 seconds and an interest window of 8 seconds. To evaluate the quality of each auditing strategy, we evaluate the average download factors of correct nodes during a 100 second time interval after auditing is first applied to the system. For the sample-based techniques, we considered that global auditors collected information from 100 nodes between each interval of 20 seconds. Notice that the sample size does not increase with the size of the system, which is a positive aspect of the auditing approach. In Subsection 4.4.1, we discuss the costs involved in collecting these samples.

In Figure 4.8, we consider the use of fixed thresholds. We studied the effects of using different values for t, starting from 0 (no auditing) and increasing it until 0.9 (90% of the stream rate), and present a detailed set of results on applying different thresholds to different scenarios. In each scenario, 30% of all nodes are freeloaders. Among freeloaders, contribution rates were set to one of 0, 100, 200, or 300 Kbps (each freeloader was randomly assigned one of these contribution rates at the beginning of a session). All other 70% nodes follow the protocol, with a maximum contribution rate set to 550 Kbps (upload factor = 1.1). We present the average download factors (Figure 4.8(a)) and the number of correct nodes mistakenly removed from the system, termed false positives (Figure 4.8(b)), for each of these configurations. The threshold applied is presented



(a) Average Download



(b) False Positives

Figure 4.8: Fixed-threshold strategy

on the x-axis. In Figure 4.8(a), as the threshold increases, higher download averages are observed, since more freeloading nodes are detected and punished. However, the number of nodes incorrectly accused also increases with higher thresholds, as observed in Figure 4.8(b).

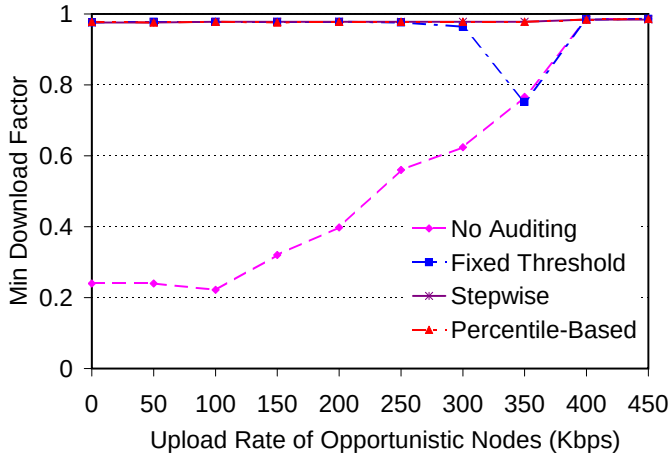
Scenarios where freeloading nodes contribute at higher rates (300 Kbps) are less disruptive to the system, but they also require higher thresholds to be applied. Different thresholds yield best results under different scenarios, but overall, from the results presented in Figure 4.8, we concluded that the best fixed

Table 4.1: Strategies for defining the threshold t

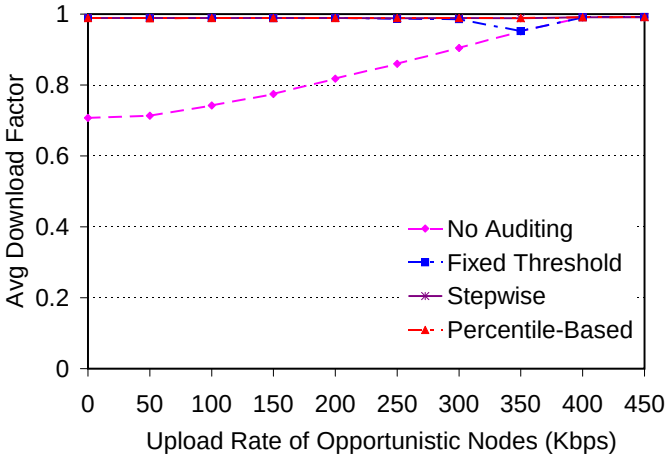
Strategy	**Description**
No auditing	Fixed t = 0.0
Fixed-threshold	Fixed t = 0.6
Stepwise adaptive	Minimum t = 0.6. If avg sampled download factor $<$ 0.97, increase t by 0.1. Decrease t back to 0.6 when avg download is satisfactory again.
Percentile-based adaptive	Minimum t = 0.0. If avg sampled download factor $<$ 0.97, t is chosen based on sampled upload factors (t = 10th percentile of sampled upload values).

threshold is $t = 0.6$, providing the best compromise in terms of performance and false positives across all scenarios.

In Figure 4.9, we compare all three strategies presented in subsection 4.3.2 against each other and against a configuration with no auditing, under different scenarios. We set t = 0.6 for the fixed threshold strategy and as the initial threshold in the *stepwise adaptive* strategy. We summarize the three strategies in Table 4.1. We simulated sessions where 30% of the nodes were freeloaders and with varying ratios of contribution. In the x-axis, the contribution rate of freeloading nodes is varied from 0 to 450 Kbps. All other nodes are correct, contributing at a maximum rate of 550 Kbps. We present both the average and the minimum download factors across all correct nodes in the system. As the contribution rate of freeloading nodes increases, the download factors are expected to increase, which is clear from the curves presented.



(a) Minimum Download



(b) Average Download

Figure 4.9: Effect of strategies when freeloading rate is varied

Figure 4.9 shows that all strategies yield significantly better results compared to an approach with no auditing. While both adaptive strategies yield excellent download rates to correct nodes, the fixed-threshold strategy's performance is not as good when freeloading nodes are contributing with 300 or slightly more Kbps (near 0.6 contribution factor). At those rates freeloading nodes are harmful to the system, yet a threshold of 0.6 is not able to detect them.

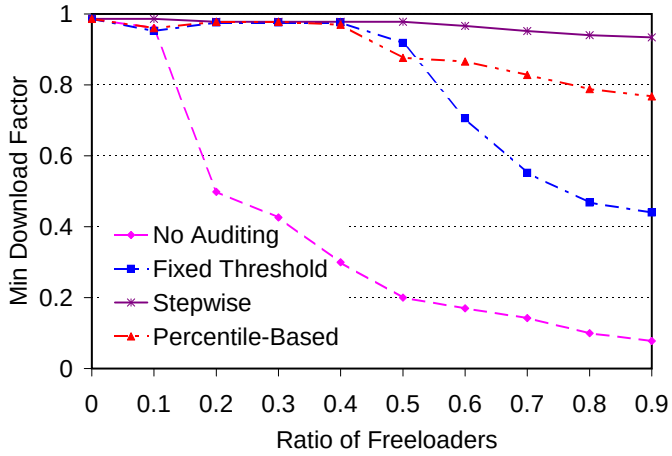
Finally, in Figure 4.10, we consider a scenario where freeloading nodes contribute with different rates. We varied the percentage of freeloading nodes in the

system from 0 to 90%, and evenly assigned them different contribution rates. The graphs present the average and minimum download rates for these scenarios. Once again, no auditing performs significantly worse than any of the auditing strategies. Here, the stepwise adaptive approach yields the best results when large percentages of freeloading nodes are present in the system. It is also simpler than the percentile-based approach, since it is based only on samples of the download rates of nodes. In both sets of experiments, the number of false positives was practically null under all three strategies considered (at most one in some cases).

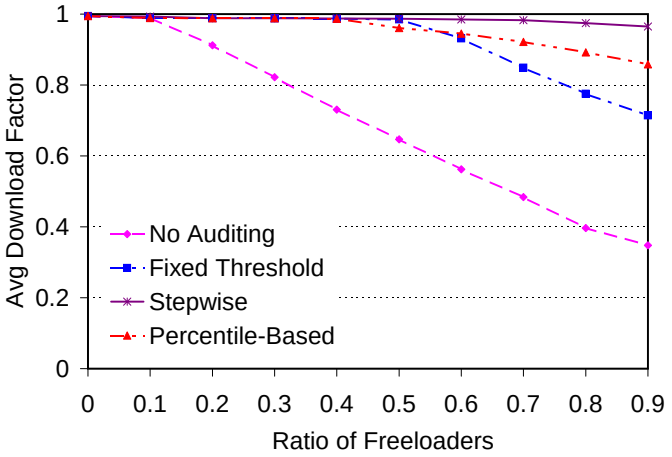
4.4.1 Auditing Costs

The overheads imposed by auditing are an important consideration, which we address in this subsection. Most of the work of auditing is performed by local auditors, which are executed on the user nodes. Overheads include local storage used to store the history information of the node, bandwidth required to exchange histories, and computation overhead in verifying other nodes' histories. We were mostly concerned with bandwidth costs since they are limited in P2P systems.

These overheads are constant, independent of the size of the system, and not significant, since nodes only exchange a small amount of accounting data at predefined intervals of time (for example, 10 seconds). If we consider a packet rate of 50 packets/s, in 10 seconds the maximum number of packets received and sent by each node is 1000. For each packet sent or received, the history needs to indicate which neighbor sent or received the packet. By using 4 bits to identify



(a) Minimum Download



(b) Average Download

Figure 4.10: Effect of strategies when percentage of freeloaders is varied

each neighbor, the size of each history packet adds up to 4000 bits (500 Bytes). This is not significant compared to the amount of regular data exchanged in a streaming session.

We also analyzed the costs of the global auditors. Since they are dedicated and external to the system, the overhead imposed on them is of higher concern. Global auditors' main tasks consist of sampling the system to collect download and upload rates of nodes, and of occasionally disseminating updates to the threshold value, through gossip. The sample size remains fixed independent

of the size of the population. We ran simulations to estimate the worst-case standard deviation of the download rates across all nodes. Accordingly, we estimate that a sample size of 300 nodes is sufficient to provide 95% confidence, independent of the population size. For smaller systems, such as the ones simulated in this work, even a smaller number of samples was found to be sufficient to yield satisfactory results. Therefore, centralized costs are fixed, and provide a clear advantage for using auditing against tit-for-tat approaches in large-scale systems.

4.5 Summary

In this chapter we presented and evaluated a scalable auditing-based technique for enforcing fairness in a live-streaming system. Our approach employs local auditors that execute on all nodes in a streaming session. They are responsible for collecting auditable information about other neighbors' data exchanges, and for verifying that neighbors upload more data than a specified threshold. This threshold is defined by dedicated global auditors, which periodically sample the state of the system to determine if the overall download rate is compromised by the presence of opportunistic nodes. Global auditing determines the minimum threshold for uploads, and works with local auditing to punish nodes that do not upload enough data.

We evaluated the efficiency of our auditing approach through simulation. We compared different strategies for establishing the contribution threshold and showed that a simple dynamic mechanism is able to maintain the throughput

of the streaming system even in the presence of a large number of opportunistic nodes.

CHAPTER 5

CARAMBOLA: HETEROGENEITY-AWARE LIVE STREAMING

5.1 Introduction

Several peer-to-peer (P2P) protocols, and in particular the majority of live streaming ones, characterize nodes as having homogeneous bandwidth, which unfortunately is not realistic. Nodes often have asymmetric upload and download rates and cannot contribute with as much bandwidth as ideally expected. Meanwhile, some nodes have higher upload bandwidth, being able to contribute more to the dissemination, if so desired. In this chapter, we study the effect of heterogeneous upload bandwidths in live streaming systems and study the potential of applying a simple placement technique for addressing problems originating from heterogeneity.

We argue for fairness in the dissemination process, suggesting that nodes that contribute with more data should also be granted better quality of received data. Meanwhile, nodes that are not able or unwilling to contribute enough upload bandwidth may receive data at a lower rate (at the highest rate allowed by the remaining upload resources). Throughout this chapter, the term heterogeneity is used in reference to nodes' upload bandwidth, given that it is one of the properties that most significantly affects P2P live streaming protocols.

To illustrate the problem, in Figure 5.1 we present an example of a heterogeneous system containing nodes with two different upload rates: 300 Kbps and 1 Mbps. The upload rate of each node is presented close to each node in the Figure. In this simple example, the circled node may not be able to receive high-

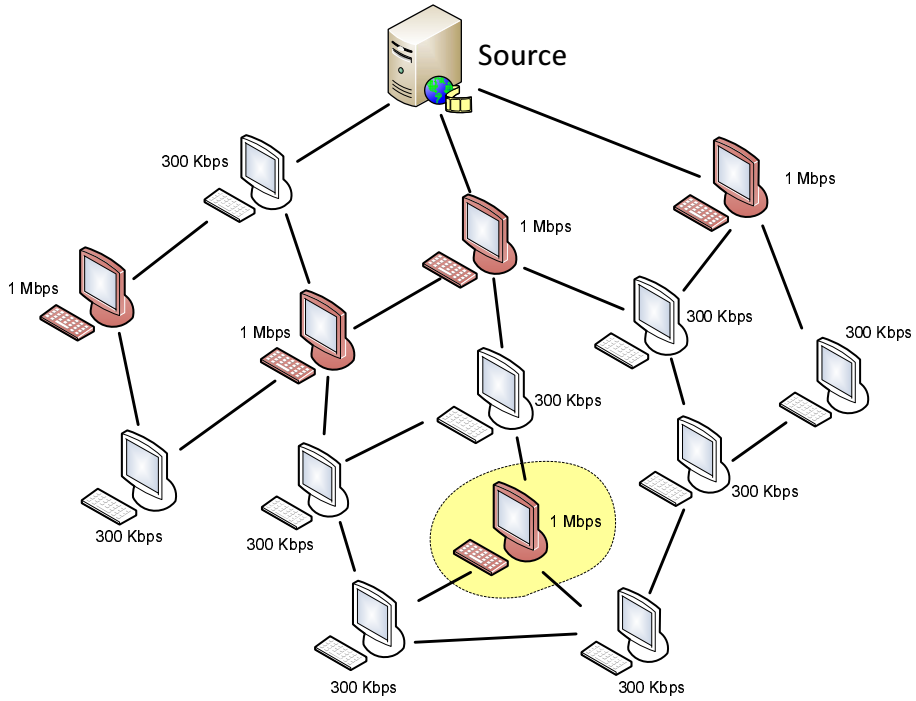


Figure 5.1: Overlay with nodes with heterogeneous bandwidths

quality data, given that its download rate is limited by the upload bandwidth of its neighbors, all of which have limited resources. Furthermore, this configuration may also prevent nodes with high upload bandwidth from generously contributing to the system at their full capacity.

Our goal is to explore node placement as well as multi-layered video coding techniques to achieve a scalable and adaptive substrate for live streaming applications. We explore different node placement techniques and analyze them based on metrics such as fairness and quality of data delivery. We expect to honor fairness by proposing a scheme that delivers data to users with quality proportional to their contributed upload capacity. Our approach, *Carambola*, organizes nodes into hierarchical layers according to their upload capacities: higher-level layers receive and propagate better quality data to other members of its layer and filter data that is propagated to members of lower-level layers.

We evaluate our approach through simulation. Our experimental results indicate that Carambola allows fair service quality metrics to be established and respected according to users' upload bandwidth. Carambola is able to satisfy our defined set of stream quality metrics, allowing the system to adapt according to receivers' bandwidth restrictions, providing the best possible stream quality for each profile.

Effects of Heterogeneity

To quantify the effects of node heterogeneity in mesh-based protocols, we ran experiments using an event-based simulator of the Chainsaw protocol [76]. Two node profiles were defined for the experiments, namely *slow* and *fast*. Slow nodes were able to upload 384 Kbps, while the upload capacity of fast nodes was varied from approximately 1.0 to 3.0 Mbps. The speeds chosen for fast nodes correspond to multiples of the upload capacity of slow nodes: approximately 2.5, 5.0 and 7.5 times higher. The upload capacity of the source of the stream was fixed at 10 times the stream rate, and the download bandwidth of nodes was set to a value higher than the stream rate, sufficient for all nodes to receive all data.

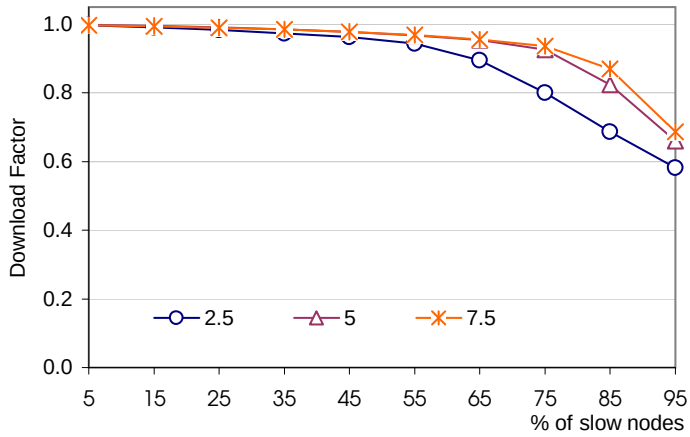
A particular *configuration* is defined by fixed values for the stream rate, and fixed upload bandwidths for slow and fast nodes. In our studies, we considered 6 different configurations, which are presented in Table 5.1. Streaming sessions with rates of 600 Kbps and 850 Kbps were considered. In all simulations, results were averaged across 100 runs of a streaming session, and node topologies were reconstructed between successive runs.

Table 5.1: Configurations defining streaming rate and upload rates of fast and slow nodes

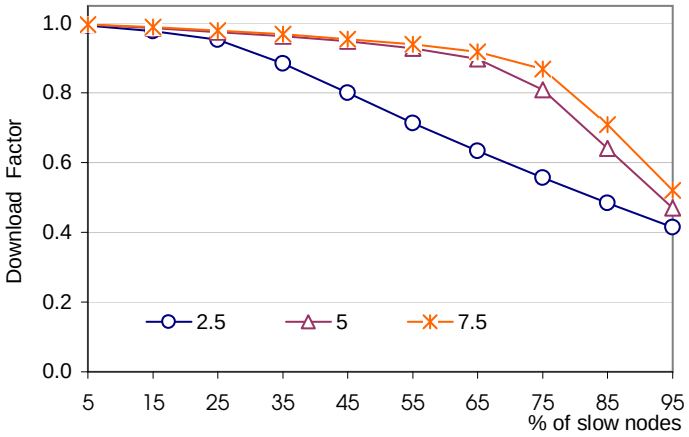
Config. Label	Stream Rate	Upload BW		Trans. Point
		Slow	Fast	(% slow nodes)
2.5 - 600 Kbps	600 Kbps	384 Kbps	960 Kbps	63%
5.0 - 600 Kbps			1920 Kbps	86%
7.5 - 600 Kbps			2880 Kbps	91%
2.5 - 850 Kbps	850 Kbps	384 Kbps	960 Kbps	28%
5.0 - 850 Kbps			1920 Kbps	73%
7.5 - 850 Kbps			2880 Kbps	83%

Notice that configurations do not define the proportion of fast and slow nodes in the system. The proportion of fast and slow nodes, given a particular configuration, defines whether the system is *poor*, *balanced* or *rich* regarding overall upload bandwidth. In a *poor* setting, the average upload bandwidth across all nodes is smaller than the stream rate, not allowing all nodes to receive 100% of the data. In a *balanced* configuration, the average upload capacity is equal to the stream rate, and in a *rich* configuration the average upload bandwidth is higher than the stream rate, potentially allowing all nodes to receive all data. We define the *transition point* of each configuration as the percentage of slow nodes for which the system is in a balanced state. Given a particular configuration, any percentage of slow nodes less than that defined by its transition point can potentially provide all nodes with 100% of the streamed data.

In our first set of experiments, nodes were organized into a randomly generated fully connected mesh. We first studied how the ratio of slow nodes af-



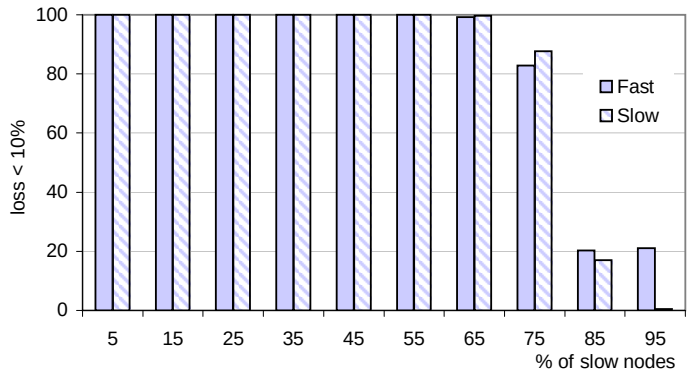
(a) 600 Kbps stream rate



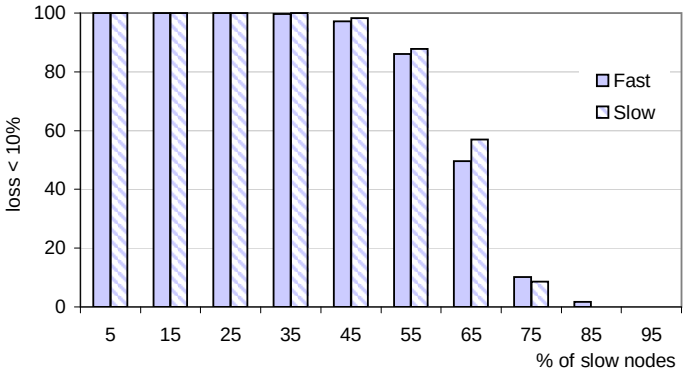
(b) 850 Kbps stream rate

Figure 5.2: Download rates when using random placement

fected the overall data distribution in the system. We varied the percentage of slow nodes from 5% to 95%, with increments of 10%. The graphs in Figure 5.2 present the results when streaming 600 and 850 Kbps, respectively, using three different profiles of upload capacity of fast nodes in each (2.5, 5.0 and 7.5 times slow nodes' upload rate). The metric we present is the *average download factor* of nodes. It captures the average ratio of streamed packets that are successfully received by each node relative to the stream rate. Each curve corresponds to a particular configuration defined in Table 5.1.



(a) 600 Kbps stream rate



(b) 850 Kbps stream rate

Figure 5.3: Percentage of nodes that receive over 90% of data

As expected, increasing the ratio of slow nodes reduces the quality of streams. Depending on the percentage of slow nodes, the overall upload capacity of the network is sometimes lower than the minimum needed to distribute the contents to all potential receivers. For a stream rate of 600 Kbps, the transition points are 63%, 86% and 91% for the three configurations presented in Figure 5.2(a). However, the quality of the dissemination becomes unsatisfactory even before the system is in a balanced or scarce state. The same can be observed in the results with a stream rate of 850 Kbps, presented in Figure 5.2(b) (for which transition points are 28%, 73% and 83%).

In Figure 5.3, we study the 5.0 - 600 Kbps and 5.0 - 850 Kbps configurations in further detail (upload bandwidth of fast nodes is five times faster than that of slow nodes). We present the percentage of fast and slow nodes that receive over 90% of the packets within their expiration time, again varying the percentage of slow nodes from 5% to 95%. We can observe that when the system's overall upload capacity is reduced (caused by larger ratios of slow nodes), fast nodes are not guaranteed to receive all data, despite their higher levels of contribution. In some cases, nodes contributing less receive more data than nodes willing to contribute with high upload bandwidths.

In realistic scenarios, we expect nodes with lower upload bandwidth to be more common than better provisioned nodes. For comparison purposes, from hereon we will work with configurations composed of 25% fast nodes and 75% slow nodes. The losses experienced by nodes in randomly organized mesh structures highlight the importance of exploring placement properties in live streaming systems.

5.2 System Design

In this section, we present Carambola, our topological approach to leverage bandwidth heterogeneity in live streaming systems. Our goal with Carambola is to offer a variable flow of packets to heterogeneous nodes, where each node's rate of received packets is proportional to its contributed upload capacity. We begin by presenting the system model and our basic assumptions. Later, we describe the basic protocol, followed by experimental results obtained through

simulation. We then explore and evaluate the use of packet filtering, and present a study of packet latency when using Carambola.

5.2.1 System Model and Assumptions

Our model assumes the existence of one source and a fixed set of consumers interested in receiving the streamed data within a specific fixed delay. The source has limited upload resources, which allows it to send data only to a small subset of consumers. The original stream consists of a sequence of packets, which may be sent out of order and reassembled by the receivers. Depending on the method used to code packets, some of them may be neglected [14, 47]. Packets may include redundant information for error correction or coding schemes that may decompose data into hierarchical information levels.

Our approach assumes that the upload capacity of nodes interested in the stream can be determined when they initially join the network. This knowledge may be provided by the node itself, or derived through system evaluation, and is required since we use the upload capacity as a reference of the node's ability to contribute to the network. Although nodes often present asymmetric download/upload rates, we employ a homogeneous exchange: a node's declared upload parameter is used to establish the target quality of data received by the node. We again use the qualifiers *fast* and *slow* to refer to nodes with higher and smaller upload capacities.

All nodes are assumed to have at least as much download capacity as the stream rate. Among broadband users, which is the class of users we consider in our studies, this is a realistic assumption. Download rates are typically higher

than upload rates in asymmetric links, and download rates are often sufficient for typical video transmissions.

We also assume that nodes follow the protocol as defined, contributing as much to the network as they declare when joining the system. The auditing techniques presented in Chapter 4 may be employed in untrusted settings to enforce that nodes contribute as much as they are expected to.

5.2.2 Basic protocol

In Carambola, nodes are organized into multiple hierarchical layers based on their upload capacities: faster nodes are placed in higher layers, closer to the source, while slower ones are placed progressively farther away. Nodes within each layer are organized into a connected mesh, in which nodes have approximately the same number of neighbors. The source is only connected to a subset of nodes from the topmost layer. Furthermore, unidirectional links are maintained between nodes in different layers, with the goal of boosting the propagation of packets down the hierarchy. All fast nodes have one or more slower nodes as neighbors, to which they forward packets.

An example of an overlay with 5 fast nodes and 15 slow nodes under the Carambola configuration is presented in Figure 5.4. In the example, each fast node is connected to two slow nodes (we refer to this configuration as 1p2). Although we only consider two node profiles (fast and slow), Carambola may easily be extended to larger numbers of layers.

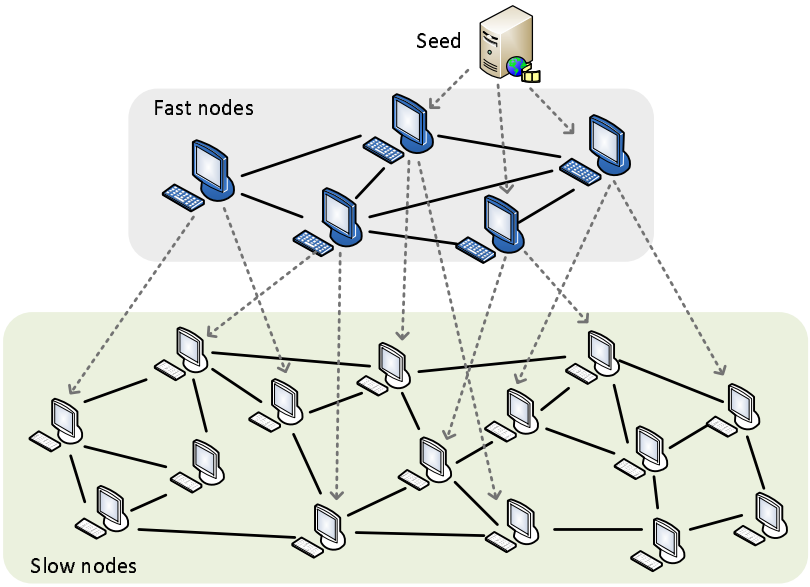


Figure 5.4: Multi-layer node configuration.

Nodes in the highest layer aim to receive all the data, as originally sent by the source. Within each layer, a pull-based style of packet exchange is employed, similar to the Chainsaw protocol [76]. Peers notify their neighbors whenever new packets are received, and request packets based on the received notifications.

Links between nodes of adjacent layers are used to disseminate data down the hierarchy, and help reduce the delay with which slower nodes receive data. This happens because fast nodes notify the receipt of new packets to its fast and slow neighbors simultaneously, allowing slow nodes to request data from fast nodes as soon as fast nodes receive them from the source. This keeps the average packet arrival delays at slow nodes close to the delays observed on fast nodes. These links also allow slow nodes to receive more data than they would be able to, given that it leverages the extra upload capacity of faster nodes.

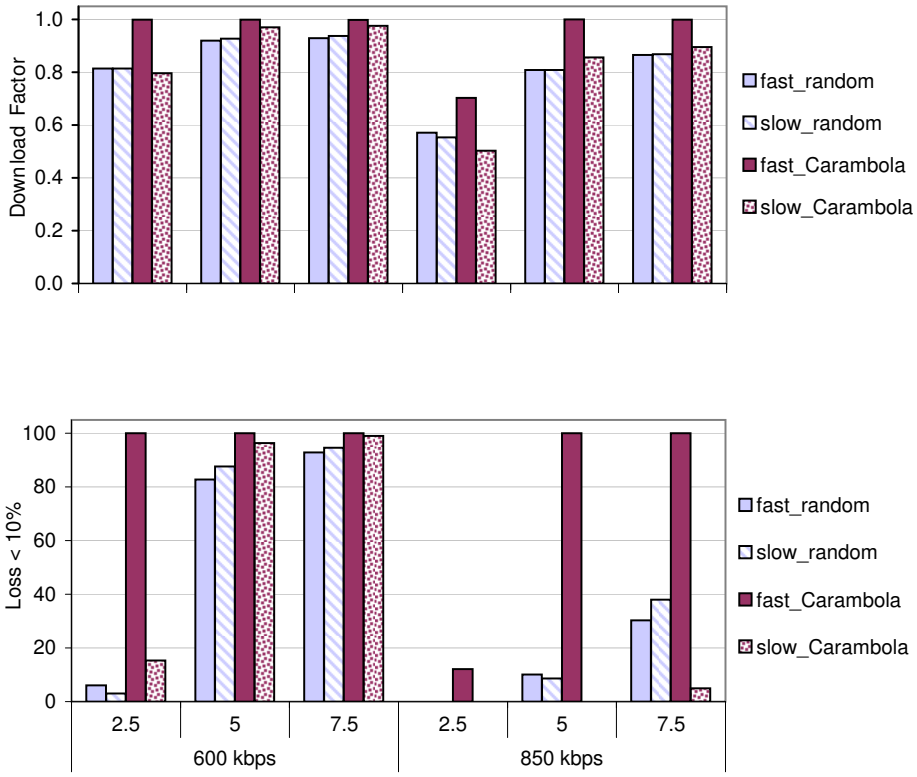
For nodes in the lower hierarchy, nodes in the higher hierarchy mimic the behavior of the source. There is one particular difference with respect to links connecting nodes of different layers: the flow of data is unidirectional, and packets are always sent from nodes in faster layers towards nodes in slower layers. Furthermore, if the aggregated upload capacity of the system is not enough to serve all nodes with all data, not all packets are notified and forwarded to slower layers, but instead, only data that has undergone a filtering process.

This approach presents two main advantages. First, it leads to a fair system, where nodes that contribute with more bandwidth resources are awarded with higher quality streams, while nodes contributing less receive lower quality streams. Furthermore, exceeding upload capacity of fast nodes may be explored to provide slower nodes with higher quality data.

5.3 Evaluation

We modified the simulator to evaluate the effect of using our approach to stream data. For comparison purposes, the configurations previously defined in the previous section were used in the experiments. Given these configurations, in this section we focus on a fixed ratio of 25% fast nodes and 75% slow nodes. In Figure 5.5(a), we present the average download factor of fast and slow nodes during sessions using (a) random node placement and (b) the placement performed by Carambola. Each cluster of columns refers to a particular configuration.

From the graph, it can be observed that when employing a random node placement strategy, the average download rates of fast and slow nodes are sim-



(b) Percentage of nodes that receive over 90% of data

Figure 5.5: Comparison between random and Carambola topologies

ilar. In some cases, slow nodes are unfairly privileged over fast nodes. With Carambola, a different pattern is observed: fast nodes, which are closer to the source and only limited by their own upload capacities, achieve satisfactory results in most cases. In all configurations, our approach eliminates the constraints less provisioned neighbors impose on fast nodes. Fast nodes are favored, being allowed to first exchange packets amongst each other, before leveraging their extra upload capacity to provide better quality streams to slower nodes.

Figure 5.5(b) presents the percentage of nodes with packet loss $< 10\%$ for the same set of experiments. For all configurations, Carambola outperforms the random placement approach, and in all but the scarcest configuration ($2.5 - 850$

Kbps), most fast nodes receive over 90% of the data. In two of the richer configurations (5.0 − 600 Kbps and 7.5 − 600 Kbps) most slow nodes also receive over 90% of data, leveraging resources from fast nodes.

The unidirectional links between fast and slow nodes may benefit slow nodes, and by doing so may harm the receipt of data by fast nodes in scarce configurations. This can be observed from the bars relative to configuration 2.5 − 850 Kbps. In this set of experiments, each fast node was linked to three slow nodes, with no particular control mechanism in place to avoid slow nodes from pulling too much bandwidth from fast nodes. In the next subsection, we explore the effect of varying parameters relative to these inter-layer connections.

5.3.1 Inter-layer links

Carambola's basic protocol defines unidirectional links between successive layers to ensure flow of data to less privileged layers. In all previous experiments, each fast node was connected to three slow nodes. However, the number of links per fast node may be varied, and the ideal value depends on factors that define the scarcity of the system, such as the ratio of fast to slow nodes, the upload capacity of the nodes, and the target streaming rate.

In Figure 5.6, we compare the quality of streaming when using 3, 2, and 1 links per fast node, respectively. The graph presents the average download factor across all nodes for each configuration. Notice that with a composition consisting of 25% fast and 75% slow nodes, in order for every slow node to be connected to at least one fast node, the number of inter-layer links per fast node should be at least 3; only configuration 1p3 satisfies this requirement. This

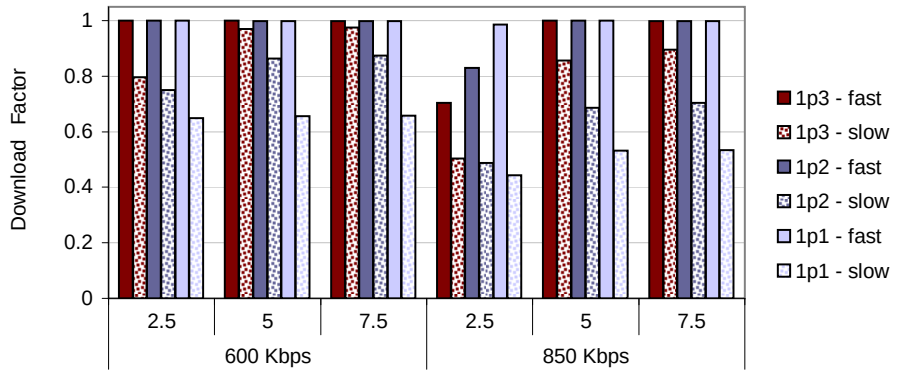


Figure 5.6: Effect of number of links between layers

property is desired when fast nodes are over-provisioned and slow nodes can benefit from fast nodes' excess of upload capacity. However, as observed in the bars for configuration $2.5 - 850$ Kbps in Figure 5.5(b), unlimited access to resources at fast nodes can harm the exchange of packets amongst fast nodes.

When we reduce the number of slow nodes connected to each fast node (1p1 and 1p2), two situations may occur: (1) in scarce configurations, fast nodes' resources are not aggressively consumed by slow nodes, and therefore, their upload capacity is first used to disseminate data to their similar neighbors and improve the results of their own layer, before providing data to their less provisioned neighbors; (2) in rich configurations, the excess upload bandwidth may not entirely be used to supply data to slow nodes.

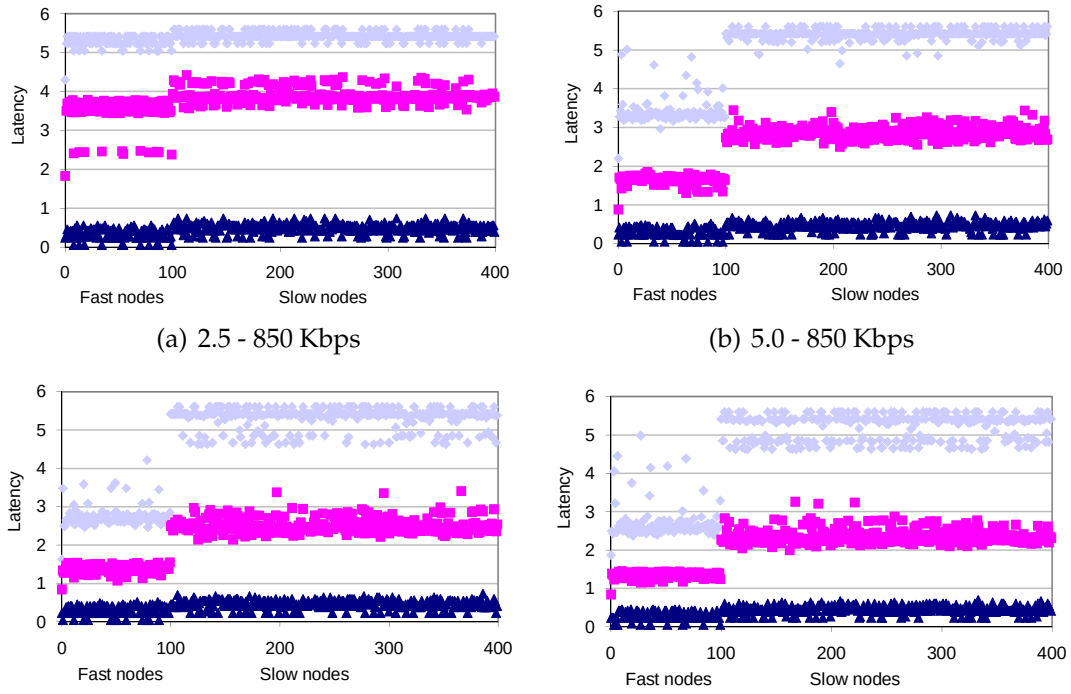
We can observe these two effects in Figure 5.6. In configuration $2.5 - 850$ Kbps, which is a scarce configuration, using less connections between fast and slow nodes reduces the quality of streams to slow nodes while it improves that of fast nodes. Meanwhile, in the remaining configurations all fast nodes receive full data but slow nodes are affected by the reduction in the number of connections, leading to underutilization of resources.

The node placement approach explored by Carambola establishes clear boundaries between layers of nodes with different bandwidth characteristics. At these boundaries, data may be filtered before propagated to nodes in lower layers, an alternative that may be explored in scarce configurations, where not enough upload bandwidth is available to provide all nodes with 100% of data. The amount of filtering reduction may be tuned depending on the configuration of the system, specially the upload capacity of slow nodes.

The MPEG video-coding technology favors the use of filtering because it allows data to be organized in layers [71]. The separate sub-streams representing individual layers are called base and enhancement layers. The *base layer* is independently coded and decoded, not needing any additional information. Each following layer (enhancement layer) is hierarchically coded based on the previous one. When incorporating filtering to Carambola, fast nodes receive both the base and enhancement layers, but only forward the base layer to slow nodes. Filtering requires fast nodes to examine the header of packets to identify if they belong to the base or enhancement layers, and only send notifications of base layer packets to slower neighbors. When more than two layers are used for grouping nodes with different upload capacities, multiple enhancement layers may be used.

5.3.2 Packet Latency

We evaluated the effect of organizing nodes into interconnected hierarchical layers on packet latencies. We measured the delays of packets received by each individual node in streaming sessions containing 100 fast nodes and 300 slow



(a) 2.5 - 850 Kbps

(b) 5.0 - 850 Kbps

(c) 7.5 - 850 Kbps

(d) 10.0 - 850 Kbps

Figure 5.7: Latency in packet distribution without filtering.

nodes, with each fast node connected to three slow nodes. We considered four upload capacities for fast nodes, namely 2.5, 5.0, 7.5, and 10.0 times higher than the upload capacity of slow nodes, which was fixed at 384 Kbps. The target streaming rate was fixed at 850 Kbps.

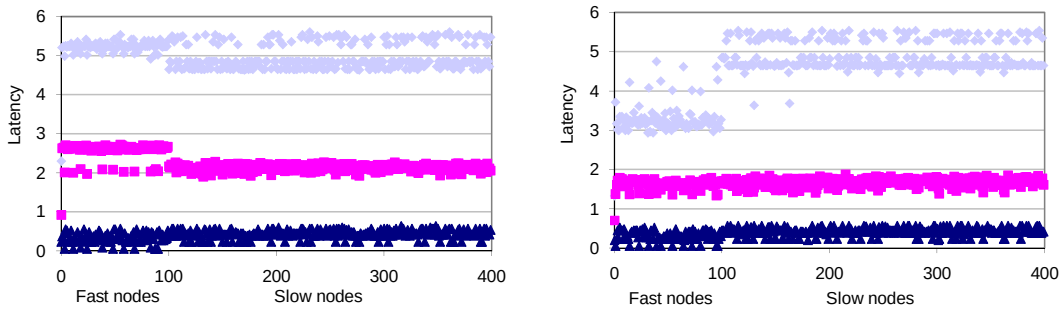
Scatter graphs presenting the minimum, average and maximum latencies (in seconds) observed by each node in sample streaming session are presented in Figure 5.7. Fast nodes may be easily identified in the leftmost one quarter of each graph (nodes 1 to 100), while slow nodes' latencies may be visualized in the rightmost three quarters of each graph (nodes 101 to 400). The experiments were also repeated with 5,000 nodes with no significant differences in latency results. No filtering between layers was applied.

It can be observed from the graphs that, as expected, the latencies of slow nodes are higher than that experienced by fast nodes, even though their average values are not significantly different. The average and maximum delays improve between graphs 5.7(a) and 5.7(b), where the upload capacity of fast nodes increases from 2.5 to 5.0 times that of slow nodes. These first two configurations are scarce, which explains the observed differences in packet latencies. When the system is under-provisioned, fast nodes' resources are in higher demand, and therefore, packet distribution among fast nodes is slowed down. Once the configuration becomes balanced, latencies do not improve much further: the rich configuration presented in Figure 5.7(d) does not present significantly improved latencies compared to the balanced configuration in Figure 5.7(c).

We also considered the latency of packets when filtering is applied. Since less data is propagated to slower nodes, we expected latencies to be lower than in unfiltered systems. In Figure 5.8, we present latencies for configurations 2.5 - 850 Kbps and 5.0 - 850 Kbps with 50% filtering between layers. When filtering is applied, reductions in packet latency are observed mainly for slow nodes. The average packet latencies are practically the same for fast and slow nodes, unlike scenarios with no filtering. Maximum packet latency across nodes is still higher for slow nodes.

5.4 Summary

In this chapter we studied the effect of peers' heterogeneous upload bandwidths on the quality of live streaming applications and presented Carambola, a new approach for organizing nodes that leads to increased fairness in the distribu-



(a) 2.5 - 850 Kbps (b) 5.0 - 850 Kbps

Figure 5.8: Latency in packet distribution with filtering.

tion of resources. Nodes in Carambola are organized into hierarchical mesh-based layers, where layers containing nodes with higher upload bandwidths are logically placed closer to the source of the stream. The advantages of mesh-based overlays are fully explored with this approach, and the unpredictability present in approaches that rely on random node placement are eliminated. Additional connections between nodes from adjacent layers allow nodes with lower upload bandwidths to leverage the exceeding upload capacity of richer nodes while ensuring that rich nodes are guaranteed full quality data for their higher contribution. Finally, we employed data filtering across layers when the aggregate upload bandwidth of the system is not sufficient to provide all nodes with full quality data.

Through simulation, we evaluated our approach against random topologies, using various metrics, and considering several different bandwidth configurations for nodes in the system. In all cases, our results indicated that Carambola achieves better fairness, providing higher-level applications with a simple substrate that is able to disseminate data with quality proportional to nodes' levels of contribution to the system.

CHAPTER 6

NIGHTWATCH: GOSSIP-BASED DISTRIBUTION ESTIMATION

6.1 Introduction

Extending mesh-based live streaming to settings with heterogeneous bandwidths as described in the previous chapter requires nodes to organize themselves into layers based on their upload bandwidth. Furthermore, deciding whether filtering should be applied between layers, and the amount of filtering to be applied depends on the overall upload bandwidth of the system. As part of an effort to extend live streaming to heterogeneous settings, we have explored techniques that allow nodes to estimate the distribution of values held by all nodes in a system, providing them with information they can use to define parameters needed in the system.

Despite our original motivation, the knowledge of how one ranks relative to peers has broader applicability and can be put to a variety of other uses. It allows outliers to be detected, overall trends to be observed and informed predictions to be made. A tool that allows nodes to maintain an estimate of what values other peers hold for particular properties can help systems be more resilient and self-adapt in sophisticated ways. In this work, we present an approach where nodes build such estimates in a timely and scalable manner. Our approach relies on gossip-style exchange of data, and uses data synopsis techniques for minimizing the amount of data exchanged between pairs of nodes. Nodes maintain a fixed-size array of entries and periodically exchange and accumulate information obtained from other peers.

Previous work has focused on the diagnosis of individual aggregate values, such as averages, sums, minimum and maximum values of distributions. Tree-based approaches compute aggregates hierarchically and under no-failure scenarios allow exact values to be computed [99, 109]. In the presence of node failures or nodes joining and leaving the system, decentralized gossip-based techniques present a more resilient model, even though computation of exact aggregates may not always be possible [59, 54]. We do not attempt to present nodes with exact distribution models since that would lead to high costs without adding significant benefits, but instead focus on providing a more expressive model that provides nodes with an approximation of the entire distribution rather than just individual aggregates.

To restrict the storage and communication costs, we explore previous work on data synopsis from the database community, originally for approximate query answering in the context of large repositories. Typically, the goal is that within a single pass through all the data a concise representation be created which allows queries to be answered within short delays of time. Two main differences in using these techniques within our gossiping context are that: (a) a large number of duplicates are present in the sample; and (b) not all data is available to every node. We will argue in this chapter that the presence of duplicates does not significantly bias the estimated distributions, and that it is therefore simpler and more efficient to leverage their presence in the samples than attempting to remove them.

We evaluated our gossip-based approach through simulation when coupled with four data synopsis techniques. We compared these in terms of quality of the estimation, storage and bandwidth requirements, and convergence time.

All techniques were evaluated with a diverse set of distributions, including uniform, normal, heavy-tailed and bimodal distributions. By experimenting with up to 100 K nodes, we have empirically validated that a limited number of rounds and a constant message throughput per node in each round is sufficient to achieve an efficient and lightweight protocol.

6.2 System Design

6.2.1 Problem Statement

We assume a system consisting of N nodes with identifiers 1 to N. Each node i holds a numerical value x_i that measures some variable of interest to the system. The set of values X held by all nodes may follow any arbitrary distribution. The main goal of our protocol is that within a predetermined interval of time, every node is able to produce a satisfactory estimate of the distribution of values x_i held by all nodes in the system.

The set of values held by nodes may vary with time. The protocol executes in phases, which are in turn subdivided into rounds. Within each phase, each node converges to an estimate of the distribution. The estimates produced at the end of each phase are an approximation of the distribution of values held at the beginning of the phase. When a new phase is started, old values are discarded in favor of newer ones.

The notion of what is a satisfactory estimate of the distribution is subjective, and may vary depending on the purpose of the application using the estimated

distribution. Instead of attempting to achieve perfect accuracy, we focus on the best balance between space overhead and accuracy, also taking into account the time complexity of the solution.

6.2.2 Basic Protocol

Nodes execute in rounds and phases. A phase is the larger time interval in which each node produces an estimate of the distribution of values. Each phase is composed of rounds of approximately fixed duration δ (e.g. 1 second). Even though rounds have a fixed duration, strict time synchronization among nodes is not required since time is only used as a rough guideline for nodes to be aware of when to proceed to the next step of the protocol. Each node is responsible for advancing rounds based on its local clock, and advancing phases when it establishes some criteria that indicates that a phase has completed. In this subsection we focus on the steps followed by each node within a single phase.

We assume that nodes maintain a set of neighbors at any given time, by using a decentralized membership protocol such as the one presented in [6]. Each node maintains a local view (its set of neighbors), and periodically updates it by randomly picking from the local views of its neighbors and from other nodes that contact it in the previous round. Nodes always remember a list of at least as many live distinct nodes as the number of rounds in a phase (usually between 15 and 20).

Every node maintains an array of k numerical values. At the beginning of each phase, all k values in the array are set to the value x_i, originally held by the node. In each round, a node i randomly chooses a partner j and requests the

set of values stored by the partner. Once it receives the array of values from j, node i has $2k$ values, which get merged into an array of size k. In the simplest protocol, hereafter called *Swap*, merging consists of randomly picking k of these values and discarding the others.

With the *Swap* protocol, nodes randomly discard data previously available to them, therefore loosing important information when estimating the distribution of values. Data synopsis techniques allow peers to store data previously seen with limited loss of information and consume less space. We next consider three such synopsis construction techniques.

Concise Counting

The first technique we considered is an adaptation of the *counting samples* approach [40] for compressing data in large data warehouses. In the original approach, values appearing more than once in the sample are represented as a value and a count pair. Given that we are dealing with floating point numbers and have fixed storage space, the following adaptations were made: all entries in our array are tuples of <value, counter>. Whenever new values are added to the sample, the tuples are sorted based on their values, and the closest values in the sample are merged together, so that only a fixed number of tuples are in the sample at any given time. Merging two tuples consists of randomly picking one of the two values and adding their respective counters.

Equi-Width Histograms

A straightforward histogram technique breaks the range of possible values into equal sized bins, and maintaining counters for each bin. One difficulty with this *Equi-Width* approach occurs when nodes are not aware of what the extreme values of the distribution are. In our implementation, each node i initially considers the set of values to range from 0 to the value they hold (x_i), and later resizes the bins dynamically in case new values beyond the extremities are found. When resizing, each old bin is mapped to a larger new bin, based on the middle value of the old bin, and the ranges of the new resized bins. The counter of each old bin is added to the new bin to which it is mapped. The main advantage of the *Equi-Width* approach when compared to the *Concise* approach is that since bins have equal width, only the extreme values of the whole distribution and counters for each bin need to be stored, reducing the amount of data stored and transferred.

Equi-Depth Histograms

Dividing the range of values into *Equi-Width* partitions may lead to very inaccurate estimations depending on the original distribution of values. Another choice consists in using *Equi-Depth* bins, where each bin contains an approximately equal number of points. In our implementation of the *Equi-Depth* histogram approach, each node i initially divides the range $[0, x_i]$ into fixed sized bins, each represented by a pair of <value, counter>. A simple protocol is used to later merge or split bins based on their counters as new data is inserted. After exchanging data with another peer, each node orders all collected pairs of <value, counter> and computes which consecutive bins, when merged, yield

the smallest combined bin. The identified bins are merged (their counters are added and the weighted arithmetic mean of their values is used as the value of the new bin) and the process is repeated until only the desired number of bins are left. The main goal of this process is to minimize the disparity across all bins.

6.3 Evaluation

We built a round-based simulator to evaluate our gossip-based approach and to compare its behavior when coupled with the four data synopsis techniques. Unless otherwise stated, we ran experiments simulating 10.000 nodes with partial connectivity. Nodes held arrays containing 50 values, which were simultaneously updated only at the end of each round.

We used the Kolmogorov-Smirnov distance as the quality metric of a distribution estimate relative to the original distribution. The KS-distance measures the maximum vertical distance between the actual cumulative distribution and the cumulative estimated distribution. In practical terms, it measures the maximum disparity between the real and estimated percentages of nodes that hold more or less than any particular value. For instance, a KS-distance of 0.1 implies that the estimated percentage of nodes larger and smaller than some particular value might be off by up to 10%.

Therefore, the KS-distance is a general metric for evaluating the quality of the estimations when calculating percentiles. We always present the *maximum* KS-distance across all nodes in the system, which is indicative of the worst-case estimation, since we aim to achieve a protocol that allows all nodes to compute satisfactory estimates. Even though of interest, individual aggregates such as

mean, medium, min/max and others are omitted given that they are less general than the KS-distance when evaluating estimated distributions.

6.3.1 Effect of Duplicates

In our first experiment we explore the effect of duplicate values in the data samples accumulated by nodes. To estimate the time and data required in an optimal data collection scenario, we considered a non-practical protocol where nodes use gossip to exchange vectors that accumulate all data received from peers (*Gossip with Duplicates*). Nodes exchange larger arrays as rounds progress, and arrays contain duplicate values. Next, we considered a similar setting, but in which duplicates were removed (*Gossip with Duplicates Removed*). Our goal with these experiments was to analyze the penalty incurred by keeping duplicates in the collected samples.

In Figure 6.1, a comparison of the two experiments is presented over increasing numbers of rounds. In the presented example the set of values held by nodes followed an exponential distribution; the results for other distributions were similar or better and are therefore omitted. The curve for the setting where duplicates are removed shows that all nodes converge to the ideal distribution around the 15th round (the lines show the metrics for the worst-case node at any round). When duplicates are not removed, nodes converge to a maximum KS-distance of approximately 0.06, again around the 15th round. This difference in quality of the estimates is the tradeoff for the simplicity of not having to remove duplicates from the samples. All data summarization techniques we evaluate can perform at best as well as the curve for *Gossip with Duplicates*.

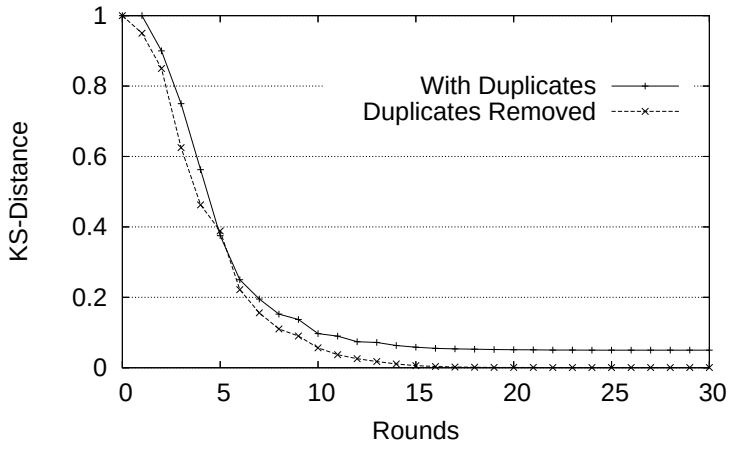


Figure 6.1: Effect of duplicates in collected samples

6.3.2 Sample Distributions

We compared the performance of the four data summarization techniques when combined with the gossip protocol in terms of quality of the estimates under a diverse set of distributions. We considered uniform, normal, exponential, Pareto, chi-square, lognormal, Weibull and multimodal distributions, all with varying parameter values. For conciseness, we only present graphs for four representative distributions: uniform, exponential ($\lambda = 1.5$), Pareto ($k = 5$, $x_m = 1$), and one bimodal distribution composed by adding two normal distributions (with parameters $\mu_1 = 5, \sigma_1 = 1$, and $\mu_2 = 8, \sigma_2 = 0.5$).

Among the distributions considered, uniform was the easiest to estimate, as observed in Figure 6.2. While the *Swap* technique falls behind with a large KS-distance, the three other summarization techniques perform well, with KS-distances always smaller than 0.1. This obviates the importance of accumulating enough values to make adequate predictions. It is worth mentioning that in the *Swap* approach some nodes are able to estimate the distribution as well as nodes in the other three approaches (not shown in the graph). However, as previously

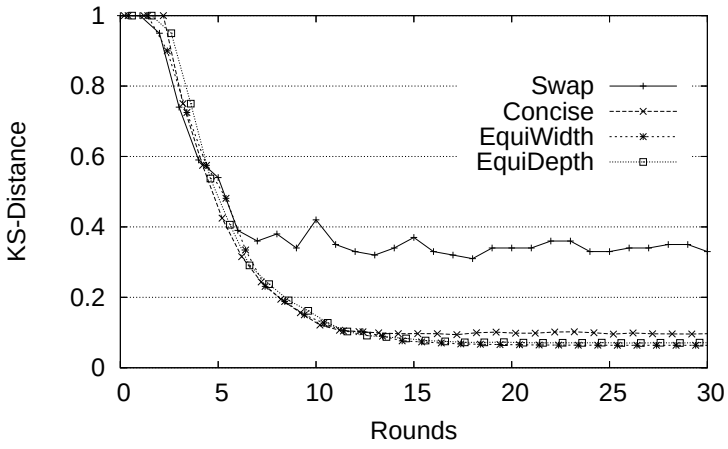


Figure 6.2: Comparison of strategies with uniform distribution

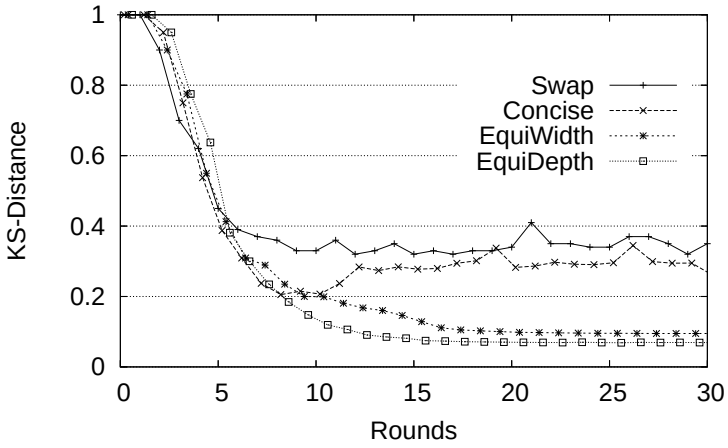


Figure 6.3: Comparison of strategies with exponential distribution

stated, we use the worst node's estimate as a metric since we expect all nodes to achieve satisfactory knowledge.

Next, we considered an exponential distribution, which as observed, affected the performance of the *Concise* technique (Figure 6.3). The main reason for this technique's poor performance is that it maintains a fixed number of <value,counter> pairs, and merges the pairs with closest values whenever needed. In distributions where data is not evenly distributed, the approach uses

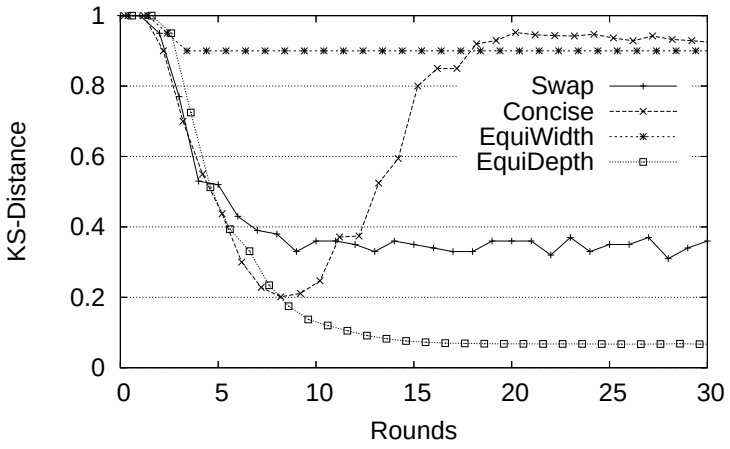


Figure 6.4: Comparison of strategies with Pareto distribution

most space storing dispersed values which represent a minority of the values in the system.

The *Equi-Width* histogram approach suffers from the same problem as the *Concise* approach since it divides the space of values into equal-sized bins. While the *Equi-Width* approach worked well with most of the distributions we considered, it failed to do so with heavy-tailed distributions such as the Pareto distribution considered for the experiment in Figure 6.4. In this distribution, most nodes hold small values that do not get differentiated into separate bins, which leads to poor computation of percentiles and large KS-distance metrics.

The bimodal distribution did not present further challenges when compared to the previous distributions despite the presence of two modes. A more detailed analysis confirmed that it is possible to compute the mean, median, and other percentiles accurately with the three later synopsis techniques.

As evidenced from the graphs for these four distributions, the *Equi-Depth* approach consistently performs well, maintaining the worst-case KS-distance metric around 0.07. This behavior was maintained when we experimented

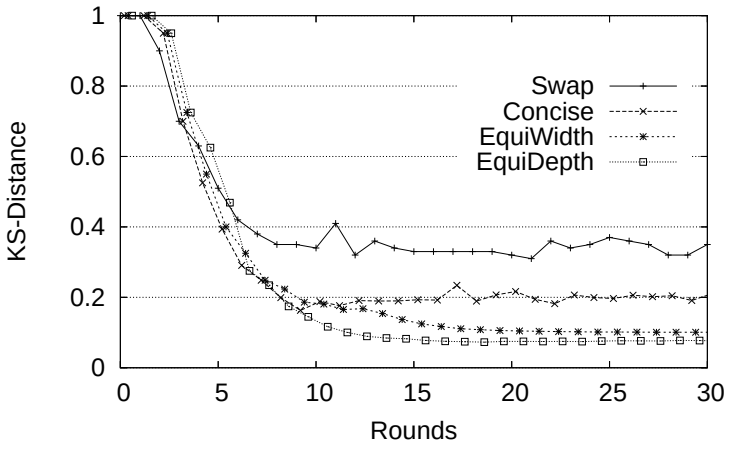
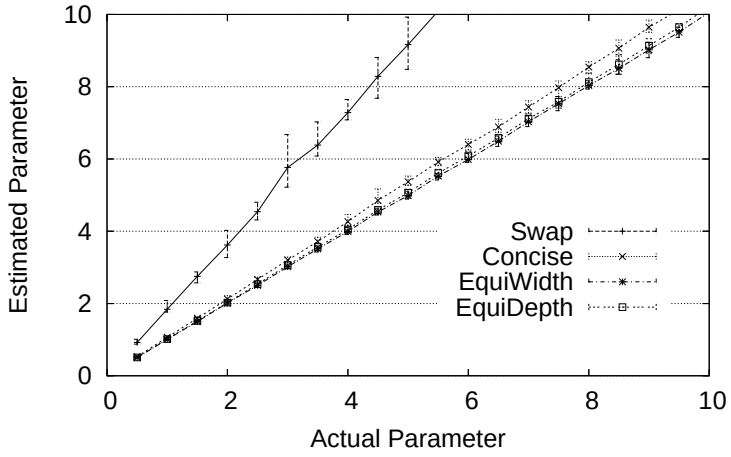


Figure 6.5: Comparison of strategies with bimodal distribution

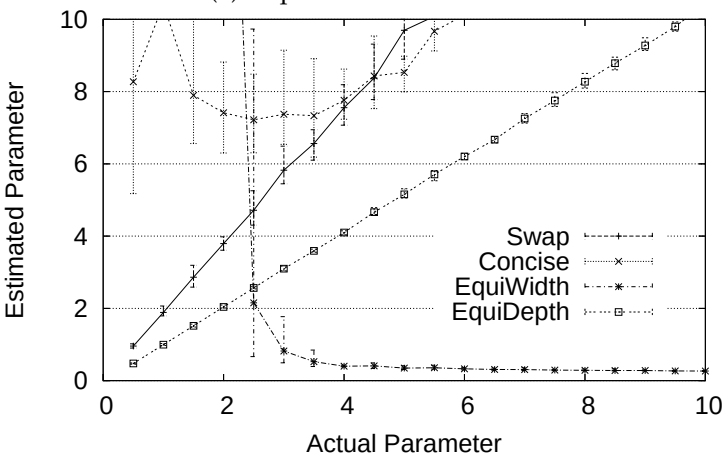
with several other distributions and parameters not presented. The *Equi-Width* technique also performed satisfactorily for most of the distributions, but as observed, significantly degrades under severely skewed distributions. The advantage of the *Equi-Width* technique lies on the fact that it requires only approximately half the space required by the *Equi-Depth* approach since only the extremity values and the bin size need to be stored.

Another approach to evaluate the different data summarization techniques consists in using the estimated distributions computed by the nodes to estimate the parameters of the original distributions (known a priori in a controlled setting). We show in Figure 6.6 how well the techniques perform in terms of estimating the parameters of exponential and Pareto distributions. On the x-axis we varied the value of the parameter used to generate the distribution of values across the nodes, and on the y-axis we present the actual worst-case estimates computed by the nodes at the end of 15 rounds.

A perfect estimation of values would be represented by the identity function. These curves validate the observations that the *Equi-Depth* approach yields the



(a) Exponential distribution



(b) Pareto distribution

Figure 6.6: Estimation of parameters

most accurate results, and shows how the *Equi-Width* and the *Concise* approach can lead to completely erroneous estimates of the original parameters of the distribution for heavy-tailed applications. Even though the *Swap* technique does not yield accurate distribution estimates, it is interesting to note that it performs consistently on all distributions.

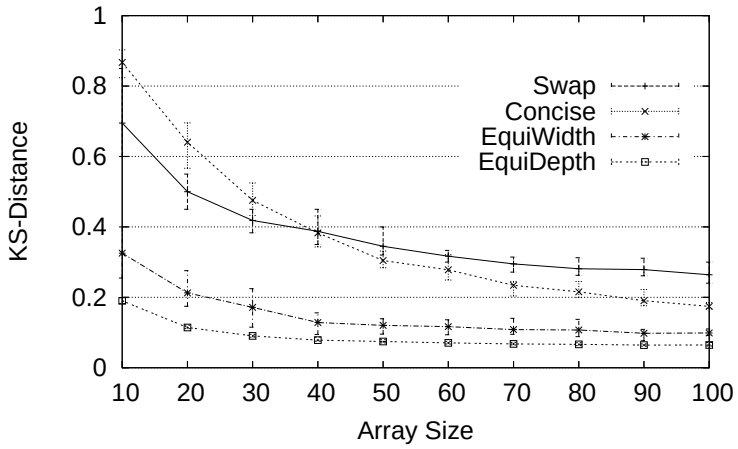
We also performed experiments with larger numbers of nodes to study how the number of rounds varies with larger systems. We experimented with up to 100 K nodes using the *Equi-Depth* technique. One interesting thing that was ob-

served was that the number of rounds for convergence remained 15, even when experimenting with 100 K nodes. While further analysis would be required to estimate the number of rounds for larger networks, the observed results are a positive indication on the time complexity of the protocol.

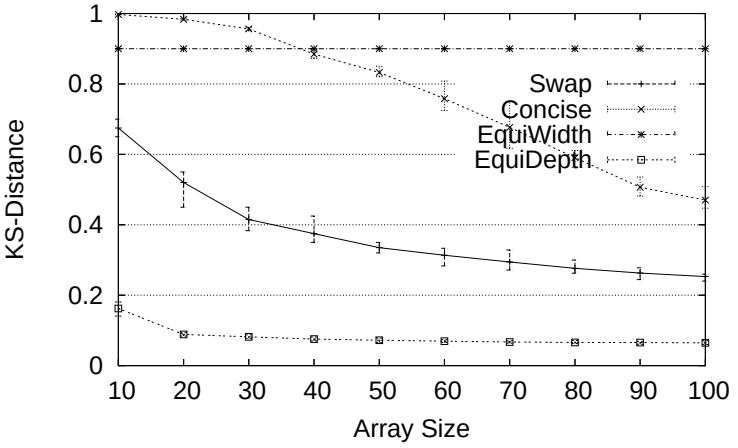
One limitation of our gossip-based approach is that although it is able to capture the general distribution of values with satisfactory accuracy, it does not necessarily reflect the effect of extreme values. This means that extreme values are not accurately recorded by nodes, and in situations where a very small number of nodes significantly affects particular aggregate values, nodes may not be able to accurately estimate these. This problem may be alleviated by pairing our approach with previously employed approaches to compute individual aggregates. Further work is required to study whether an efficient solution combining benefits of both approaches is feasible.

6.3.3 Array Size

One important parameter to consider in our approach is the size of the arrays used to store and exchange data. In all previous experiments we employed arrays with 50 elements. The types of elements in each array vary with each approach: floating point values for the *Swap* technique, integers for the *Equi-Width* approach and pairs of <float, integer> for the *Concise* and *Equi-Depth* approaches. Storage-wise, the *Swap* and *Equi-Width* approaches were more efficient in the previous experiments. To confirm that adding further storage space to these techniques would not lead to different outcomes, we evaluated the effect the array-size has on the estimations.



(a) Exponential



(b) Pareto

Figure 6.7: Effect of array size

In Figure 6.7, we present how the maximum KS-distance among nodes at the end of the 15th round varies with increasing array-sizes. We again present results for the exponential and Pareto distributions. Increasing the array-size did lead to improved results in most cases, as expected, but even with arrays of 100 elements, none of the three techniques is able to outperform the *Equi-Depth* technique with 50 elements. Another important point to notice is that the *Equi-Depth* approach does not benefit significantly from using arrays containing more than 40 or 50 elements. Depending on the bandwidth requirements of applications, even 20 or 30 elements may produce satisfactory estimates.

6.4 Summary

In this chapter we presented and evaluated NightWatch, a scalable gossip-based technique that allows nodes to estimate distributions of values held by other peers. Unlike approaches which attempt to compute individual aggregates of values, our approach aims at complementing this information with knowledge about how any value ranks relative to others. In NightWatch, nodes periodically exchange an array of data that concisely summarizes the data they have previously seen.

Through simulation, we compared different synopsis techniques for compressing data that is gossiped among nodes, thereby saving space required for storing and exchanging data. We considred several distributions in our experiments, including heavy-tailed and bimodal distributions. We observed that the data synopsis technique adopted can severely impact the quality of the estimates collected, and that the *Equi-depth* histogram technique provides a good balance between space requirements and quality of estimation.

CHAPTER 7

CONCLUSION

The Internet provides a promising medium for dissemination of multimedia contents in real time. Peer-to-peer live streaming technology has the potential to fulfill this promise by allowing any end user on the Internet to be either the producer or the consumer of data being disseminated, without requiring prohibitive networking and computational resources. Aiming for this goal, significant work has been dedicated to live streaming, and led to simple and efficient protocols. Using the upload bandwidth of the nodes interested in receiving the contents to help in the dissemination process, state of the art protocols allow data to reach thousands of receivers without requiring extra bandwidth resources at the source of the stream.

Unfortunately, practical considerations, such as security and heterogeneity of network bandwidths, were overlooked at these initial protocols. Our work makes it clear that these are important if such services are to be deployed at large scale on the Internet. A simple investigation of the properties of previous protocols exposes their vulnerabilities in the presence of a few non-cooperative nodes. In some cases, one such node is able to prevent several interested nodes from receiving any data. Furthermore, most existing protocols consider settings where all users present homogeneous upload bandwidths, which do not at all scale to settings where such assumption does not hold. This dissertation attempted to address some of these issues, and focused on techniques for improving reliability of peer-to-peer live streaming systems, and for increasing scalability to settings with heterogeneous bandwidth requirements.

7.1 Summary

This section summarizes the main contributions of this dissertation. First, the investigation of vulnerabilities of existing live streaming systems led to the design of SecureStream, a peer-to-peer live streaming protocol tailored to handle a variety of malicious attacks. This dissertation described the components of SecureStream and the main techniques employed to resist against Denial of Service, forgery, membership and omission attacks. An evaluation of SecureStream showed that the system tolerates a limited percentage of malicious nodes, gracefully degrading in the presence of increasing ratios of malicious nodes.

Next, we presented the design and evaluation of a scalable auditing-based technique for enforcing fairness in a live streaming system. The approach employs local auditors that execute on all nodes in a streaming session. They are responsible for collecting auditable information about other neighbors' data exchanges, and for verifying that neighbors upload more data than a specified threshold. This threshold is defined by dedicated global auditors, which periodically sample the state of the system to determine if the overall download rate is compromised by the presence of opportunistic nodes. Global auditing determines the minimum threshold for uploads, and works with local auditing to punish nodes that do not upload enough data. We presented results that indicate the efficiency of this auditing approach through simulation, showing that it is able to maintain the throughput of the streaming system even in the presence of a large number of opportunistic nodes.

Furthermore, we studied the effect of heterogeneous upload bandwidth on the quality of multimedia streaming applications and the design of Caram-

bola, a new approach for organizing nodes that leads to fair distribution of resources. Nodes in Carambola are organized into hierarchical mesh-based layers, with layers containing nodes with higher upload capacity being logically placed closer to the source. The advantages of mesh-based overlays are fully explored with this approach, eliminating the unpredictability present in approaches that rely on random node placement. Additional connections between nodes from adjacent layers allow nodes with less upload capacity to leverage the excess upload capacity of richer nodes while ensuring that rich nodes are guaranteed full quality data for their higher contribution. Combined with data filtering across layers, streaming is further scaled for scenarios when the aggregate upload bandwidth of the system is not sufficient to provide all nodes with full quality data. By evaluating this approach against random topologies, we showed how Carambola achieves increased fairness, providing higher-level applications with a simple substrate that is able to disseminate data with quality proportional to nodes' levels of contribution to the system.

Finally, we presented the design of a gossip-based technique that allows nodes to estimate distributions of values held by other peers. The original motivation for this work was to design a tool that allows nodes in streaming sessions to rank their bandwidths relative to other nodes in the system in a scalable and inexpensive manner. However, the tool has much wider applicability and can be used for monitoring purposes in distributed systems in general. Unlike approaches which attempt to compute individual aggregates of values, our solution aims at complementing this information with knowledge about how any value ranks relative to others. We presented a comparison of different synopsis techniques for compressing data that is gossiped among nodes, thereby saving space required for storing and exchanging data.

The techniques employed in this dissertation, although independent, were designed to complement each other, and when combined, lead to a highly resilient and scalable system. Even though further work is required, the simplicity of each technique contributes to the design of a practical and efficient service, reducing the gap towards building practical peer-to-peer live streaming systems capable of fulfilling the original goal of allowing any user to easily stream data in real time to any interested set of receivers.

7.2 Limitations and Open Questions

This section presents open research questions related to the work presented in this dissertation. Despite significant progress on P2P live streaming systems, there are still limitations and further research is required to make such systems more practical and therefore popularize the widespread use of the technology. Some possible areas of research are briefly described next.

One interesting and fairly unexplored topic of research is the consideration of network locality in the context of live streaming systems. One of the main drawbacks of application-level multicast when compared to IP multicast is the extra load on links, given that most protocols are entirely based on an overlay and ignore the underlying network topology through which the data flows. By taking peer locality into account when assigning neighbors to nodes, packet latencies and the overhead on the underlying infrastructure can potentially be reduced. DagStream [66] and Rainbow [22] are two systems which attempt to consider network locality in streaming, and present some preliminary results on the benefits of taking locality into account.

Given the vast amount of research that has been dedicated to developing live streaming protocols, research that compares existing approaches and tries to better model and understand their tradeoffs would prove very useful. Comparative studies between push and pull-based streaming protocols, for example, are an interesting topic of study that can clarify the benefits of each of these approaches. Limited work has been conducted to identify the tradeoffs between these two main paradigms, and most work has focused on protocols which follow either of them, without providing exhaustive data that show which paradigm, if any, is superior. Another interesting study consists in comparing the tit-for-tat style of contribution enforcement with audit-based approaches.

The use of auditing can also be further explored to avoid and discourage voluntary deviation from protocols. The use of authenticated messages, public auditable histories of interactions, and a combination of local and global auditing are techniques that may be generalized to other application domains. Several applications present the need for auditing/monitoring tools, and therefore one interesting topic of research consists in building a general auditing infrastructure which can be easily tuned and employed by different applications, with a user-controlled tradeoff between sensitivity of the detection system and the risk of false positives.

Finally, further work is also necessary to create data aggregation tools resilient to malicious and selfish behaviors by participating nodes. The NightWatch system does not provide any mechanisms for resisting against unexpected behavior; malicious peers may provide erroneous data to partners, radically changing the outcome of the estimated distributions. In fact, it would be challenging to modify the current protocol in order to avoid malicious behavior,

and therefore NightWatch would not provide the desired guarantees in settings where nodes are untrusted.

7.3 Concluding Remarks

This dissertation argued for the deployment of live streaming systems at the application level and attempted to enhance the state of art of peer-to-peer live streaming protocols. One of the goals consisted in understanding current vulnerabilities of existing systems and proposing techniques that alleviate or avoid the effect of attackers, while maintaining the overall performance of the system. The second main goal consisted in exploring techniques that allow streaming systems to scale to settings where nodes have heterogeneous bandwidths, allowing streaming to be deployed in more realistic settings over the Internet. While further work is still required, we hope the work presented in this dissertation has contributed towards the goal of increasing the reliability and scalability of live streaming systems.

BIBLIOGRAPHY

[1] ABC Homepage. Available: http://abc.go.com.

[2] Andrew Adams, Jonathan Nicholas, and William Siadak. Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised). RFC 3973, January 2005.

[3] Eytan Adar and Bernardo A. Huberman. Free Riding on Gnutella. Technical report, Xerox PARC, August 2000.

[4] Charu C. Aggarwal. On Biased Reservoir Sampling in the Presence of Stream Evolution. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Seoul, Korea, September 2006.

[5] Charu C. Aggarwal and Philip S. Yu. A Survey of Synopsis Construction in Data Streams. In *Data Streams: Models and Algorithms*, chapter 9. Springer-Verlag New York, LLC, 2006.

[6] André Allavena, Alan Demers, and John E. Hopcroft. Correctness of a Gossip Based Membership Protocol. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, Las Vegas, NV, July 2005.

[7] Kevin Almeroth. The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment. *IEEE Network*, 14:10–20, January 2000.

[8] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling From a Moving Window Over Streaming Data. In *Proceedings of the ACM-SIAM Symposium on Discrete algorithms (SODA)*, San Francisco, CA, January 2002.

[9] Gal Badishi, Idit Keidar, and Amir Sasson. Exposing and Eliminating Vulnerabilities to Denial of Service Attacks in Secure Gossip-Based Multicast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, Philadelphia, PA, June 2004.

[10] Anthony Ballardie, Jon Crowcroft, Christophe Diot, Cheng-Yin Lee, Radia Perlman, and Zheng Wang. On Extending the Standard IP Multicast Architecture. Technical report, University of College London, October 1999.

[11] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing. In *Proceedings of ACM SIGCOMM*, San Francisco, CA, September 1993.

[12] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, Pittsburgh, PA, October 2002.

[13] Mayank Bawa, Hector Garcia-Molina, Aristides Geonis, and Rajeev Motwani. Estimating Aggregates on a Peer-to-Peer Network. Technical report, Stanford University, 2003.

[14] Bartosz Biskupski, Raymond Cunningham, Jim Dowling, and René Meier. High-Bandwidth Mesh-based Overlay Multicast in Heterogeneous Environments. In *Proceedings of the Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA)*, Pisa, Italy, October 2006.

[15] Adrian Bozdog, Robbert van Renesse, and Dan Dumitriu. SelectCast: A Scalable and Self-Repairing Multicast Overlay Routing Facility. In *Proceedings of the ACM Workshop on Survivable and Self-Regenerative Systems (SSRS)*, Fairfax, VA, October 2003.

[16] Brad Cain, Stephen E. Deering, Isidor Kouvelas, Bill Fenner, and Ajit Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376 (Proposed Standard), October 2002.

[17] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, October 2003.

[18] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, October 2002.

[19] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate Query Processing Using Wavelets. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Cairo, Egypt, August 2000.

[20] Yatin Chawathe. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, December 2000.

[21] Yatin Chawathe. Scattercast: An Adaptable Broadcast Distribution Framework. *Multimedia Systems*, 9(1):104–118, July 2003.

[22] Yang Chen, Bei xing Deng, and Xing Li. Rainbow: A Locality-aware Peer-to-Peer Overlay Multicast System. In *Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops (GCCW)*, Hunan, China, October 2006.

[23] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A Case for End System Multicast. In *Proceedings of ACM Sigmetrics*, Santa Clara, CA, June 2000.

[24] Bram Cohen. Incentives Build Robustness in Bittorrent. In *Proceedings of the Workshop on the Economics of Peer-to-Peer Systems (P2PECON)*, Berkeley, CA, June 2003.

[25] William Conner, Klara Nahrstedt, and Indranil Gupta. Preventing DoS Attacks in Peer-to-Peer Media Streaming Systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2006.

[26] Luis Henrique M. K. Costa, Serge Fdida, and Otto Duarte. Hop by Hop Multicast Routing Protocol. In *Proceedings of ACM SIGCOMM*, San Diego, Ca, August 2001.

[27] Yogen K. Dalal and Robert M. Metcalfe. Reverse Path Forwarding of Broadcast Packets. *Communications of the ACM*, 21(12):1040–1048, December 1978.

[28] Stephen E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[29] Stephen E. Deering and David R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems (TOCS)*, 8:85–110, 1990.

[30] Stephen E. Deering, William Fenner, and Brian Haberman. Multicast Listener Discovery MLD for IPv6. RFC 2710, October 1999.

[31] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, January 2000.

[32] Hans Eriksson. MBONE: the Multicast Backbone. *Communications of the ACM*, 37(8):54–60, August 1994.

[33] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-Riding and Whitewashing in Peer-to-Peer Systems. In *Proceedings of the ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems (PINS)*, Portland, OR, August 2004.

[34] Bill Fenner, Mark Handley, Hugh Holbrook, and Isidor Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC 4601, August 2006.

[35] Bill Fenner and David Meyer. Multicast Source Discovery Protocol (MSDP). RFC 3618, October 2003.

[36] Paul Francis. Yoid: Extending the Internet Multicast Architecture. Technical report, The ICSI Center for Internet Research, April 2000.

[37] Michael J. Freedman, Ion Stoica, David Mazieres, and Scott Shenker. Group Therapy for Systems: Using Link Attestations to Manage Failures. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, February 2006.

[38] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Peer-to-Peer Membership Management for Gossip-Based Protocols. *IEEE Transactions on Computers*, 52(2):139–149, February 2003.

[39] Rosario Gennaro and Pankaj Rohatgi. How to Sign Digital Streams. In *Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, London, UK, August 1997.

[40] Phillip B. Gibbons and Yossi Matias. New Sampling-Based Summary Statistics for Improving Approximate Query Answers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, WA, June 1998.

[41] Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. Fast Incremental Maintenance of Approximate Histograms. *ACM Transactions on Database Systems*, 27(3):261–298, September 2002.

[42] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. A Measurement Study of Napster and Gnutella as Examples of Peer-to-Peer File Sharing Systems. *Multimedia Systems Journal*, 9(2):170–184, 2003.

[43] Maya Haridasan, Ingrid Jansch-Porto, and Robbert van Renesse. Enforcing Fairness in a Live-Streaming System. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2008.

[44] Maya Haridasan and Robbert van Renesse. Defense Against Intrusion in a Live Streaming Multicast System. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P)*, Cambridge, UK, September 2006.

[45] Maya Haridasan and Robbert van Renesse. SecureStream: An Intrusion-Tolerant Protocol for Live-Streaming Dissemination. *Computer Communications. Special Issue on Disruptive Networking with Peer-to-Peer Systems*, 31(3), February 2007.

[46] Maya Haridasan and Robbert van Renesse. Gossip-based Distribution Estimation in Peer-to-Peer Networks. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Tampa Bay, FL, February 2008.

[47] Barry G Haskell, Atul Puri, and Glen G. Langdon Arun N. Netravali. Digital Video: An Introduction to MPEG-2. *Journal of Electronic Imaging*, 7:265–266, January 1998.

[48] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and Keith W. Ross. Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System. In *Proceedings of the Workshop on Internet Protocol TV Services over World Wide Web (IPTV)*, Edinburgh, Scotland, May 2006.

[49] Hugh W. Holbrook and David R. Cheriton. IP Multicast Channels: EXPRESS Support for Large-Scale Single-Source Applications. In *Proceedings of ACM SIGCOMM*, New York, NY, September 1999.

[50] Piotr Indyk, Nick Koudas, and S. Muthukrishnan. Identifying Representative Trends in Massive Time Series Data Sets Using Sketches. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Cairo, Egypt, September 2000.

[51] Yannis E. Ioannidis and Viswanath Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1995.

[52] John Jamison and Rick Wilder. vBNS: the Internet Fast Lane for Research and Education. *IEEE Communications Magazine*, 35(1):60–63, January 1997.

[53] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James O'Toole Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, San Diego, CA, October 2000.

[54] Márk Jelasity, Wojtek Kovalczyk, and Maarten van Steen. An Approach to Massively Distributed Aggregate Computing on Peer-to-Peer Networks. In *Proceedings of Euromicro Conference on Parallel, Distributed and Network-Based Processing*, A Coruña, Spain, 2004.

[55] Havard Johansen, Andre Allavena, and Robbert van Renesse. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. In *Proceedings of ACM EuroSys*, Leuven, Belgium, April 2006.

[56] Seung Jun and Mustaque Ahamad. Incentives in BitTorrent Induce Free Riding. In *Proceedings of the ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, Philadelphia, PA, August 2005.

[57] Dina Katabi. The use of IP-Anycast for Building Efficient Multicast Trees. In *Proceedings of the Global Telecommunications Conference (GLOBECOM)*, Rio de Janeiro, Brazil, December 1999.

[58] Daniel Keim and Martin Heczko. Wavelets and their Applications in Databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, April 2001.

[59] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-Based Computation of Aggregate Information. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, Cambridge, MA, October 2003.

[60] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *Proceedings of the IEEE Transactions on Parallel and Distributed Systems*, 14(3), 2003.

[61] Robert P. Kooi. *The Optimization of Queries in Relational Databases*. PhD thesis, Case Western Reserver University, Cleveland, OH, September 1980.

[62] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, October 2003.

[63] Satish Kumar, Pavlin Radoslavov, David Thaler, Cengiz Alaettinoğlu, Deborah Estrin, and Mark Handley. The MASC/BGMP Architecture for Inter-Domain Multicast Routing. In *Proceedings of ACM SIGCOMM*, Vancouver, Canada, September 1998.

[64] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the Kazaa Network. In *Proceedings of the IEEE Workshop on Internet Applications (WIAPP)*, San Jose, CA, June 2003.

[65] Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. BAR Gossip. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA, 2006.

[66] Jin Liang and Klara Nahrstedt. DagStream: Locality Aware and Failure Resilient Peer-to-Peer Streaming. In *Proceedings of the Annual Multimedia Computing and Networking Conference (MMCN)*, San Jose, CA, January 2006.

[67] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation Service for Ad-hoc Sensor Networks. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[68] Nazanin Magharei and Reza Rejaie. PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, Anchorage, Alaska, April 2007.

[69] Nazanin Magharei, Reza Rejaie, and Yang Guo. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, Anchorage, Alaska, April 2007.

[70] Sara Miner and Jessica Staddon. Graph-Based Authentication of Digital Streams. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.

[71] Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, and Didier J. LeGal, editors. *MPEG Video: Compression Standard*. Digital Multimedia Standards Series. Chapman & Hall, New York, NY, 1997.

[72] John Moy. MOSPF: Analysis and Experience. RFC 1585, March 1994.

[73] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis Diffusion for Robust Aggregation in Sensor Networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SENSYS)*, Baltimore, MD, November 2004.

[74] NBC Homepage. Available: http://www.nbc.com.

[75] Tsuen-Wan Ngan, Dan S. Wallach, and Peter Druschel. Incentives-Compatible Peer-to-Peer Multicast. In *Proceedings of the Workshop on the Economics of Peer-to-Peer Systems (P2PECON)*, Cambridge, MA, June 2004.

[76] Vinay Pai, Kapil Kumar, Karthik Tamilmani, Vinay Sambamurthy, and Alexander E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Ithaca, NY, February 2005.

[77] Vinay Pai and Alexander E. Mohr. Improving Robustness of Peer-to-Peer Streaming with Incentives. In *Proceedings of the Workshop on the Economics of Networked Systems (NetEcon)*, Ann Arbor, MI, June 2006.

[78] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: an Application Level Multicast Infrastructure. In *Proceedings of the Conference on USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2001.

[79] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Song. The TESLA Broadcast Authentication Protocol. *Cryptobytes*, 5(2), 2002.

[80] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Xiaodong Song. Efficient Authentication and Signing of Multicast Streams over Lossy Channels. In *Proceedings of the IEEE Symposium on Security and Privacy*, Berkeley, CA, May 2000.

[81] Fabio Pianese, Joaquin Keller, and Ernst W. Biersack. PULSE, a Flexible P2P Live Streaming System. In *Proceedings of the IEEE Global Internet Workshop*, Barcelona, Spain, April 2006.

[82] Viswanath Poosala. *Histogram-based Estimation Techniques in Database Systems*. PhD thesis, University of Wisconsin-Maddison, Madison, WI, February 1997.

[83] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 1996.

[84] PPLive Homepage. Available: http://www.pplive.com.

[85] PPStream Homepage. Available: http://www.ppstream.com.

[86] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. Revisiting IP Multicast. In *Proceedings of ACM SIGCOMM*, Pisa, Italy, August 2006.

[87] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001.

[88] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Application-Level Multicast Using Content-Addressable Networks. In *Proceedings of the International Workshop on Networked Group Communication (NGC)*, London, United Kingdom, November 2001.

[89] Yakov Rekhter, Tony Li, and Susan Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

[90] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object address, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001.

[91] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[92] Rob Sherwood, Seungjoon Lee, and Bobby Bhattacharjee. Cooperative Peer Groups in NICE. In *Proceedings of the Conference on Computer Communications and Networking (INFOCOM)*, San Francisco, CA, April 2003.

[93] Atul Singh, Miguel Castro, Antony Rowstron, and Peter Druschel. Defending against Eclipse Attacks on Overlay Networks. In *Proceedings of the ACM SIGOPS European Workshop*, Leuven, Belgium, September 2004.

[94] Dawn Song, J. D. Tygar, and David Zuckerman. Expander Graphs for Digital Stream Authentication and Robust Overlay Networks. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.

[95] SopCast Homepage. Available: http://www.sopcast.com.

[96] David Thaler. Border Gateway Multicast Protocol (BGMP): Protocol Specification. RFC 3913, September 2004.

[97] Dave Katz Tony Bates, Ravi Chandra and Yakov Rekhter. Multiprotocol Extensions for BGP-4. RFC 4760 (Draft Standard), January 2007.

[98] D.A. Tran, K.A. Hua, and T Do. ZIGZAG: an Efficient Peer-to-Peer Scheme for Media Streaming. In *Proceedings of the Conference on Computer Communications and Networking (INFOCOM)*, San Francisco, CA, April 2003.

[99] Robbert van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.

[100] Vidhyashankar Venkataraman, Paul Francis, and John Calandrino. Chunkyspread: Multitree Unstructured Peer to Peer Multicast. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, February 2006.

[101] Jeffrey S. Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.

[102] Jeffrey Scott Vitter and Min Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, June 1999.

[103] Long Vu, Indranil Gupta, Jin Liang, and Klara Nahrstedt. Mapping the PPLive Network: Studying the Impacts of Media Streaming on P2P Overlays. Technical report, University of Illinois at Urbana-Champaign (UIUC), August 2006.

[104] David Waitzman, Craig Partridge, and Stephen E. Deering. Distance Vector Multicast Routing Protocol. RFC 1075, November 1988.

[105] Feng Wang, Yongqiang Xiong, and Jiangchuan Liu. mTreebone: mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast. In *Proceedings of the International Conference on Symposium on Distributed Computing Systems (ICDCS)*, Toronto, Canada, July 2007.

[106] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[107] Chung K. Wong and Simon S. Lam. Digital Signatures for Flows and Multicasts. *IEEE/ACM Transactions on Networking*, 7:502–513, August 1999.

[108] Tina Wong and Randy H. Katz. An Analysis of Multicast Forwarding State Scalability. In *Proceedings of the International Conference on Network Protocols (ICNP)*, Osaka, Japan, November 2000.

[109] Praveen Yalagandula and Mike Dahlin. A Scalable Distributed Information Management System. In *Proceedings of the ACM SIGCOMM*, Portland, OR, September 2004.

[110] YouTube Homepage. Available: http://www.youtube.com.

[111] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In *Proceedings of the Conference on Computer Communications and Networking (INFOCOM)*, Miami, FL, March 2005.

[112] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, January 2003.

[113] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Port Jefferson, NY, June 2001.